НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

«До захисту допущено»
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2019р.

Магістерська дисертація

зі спеціальності 121 Інженерія програмного забезпечення
за спеціалізацією Інженерія програмного забезпечення розподілених систем
на тему: Система аналізу збіжності текстової інформації для оцінки плагіату

Виконав: студент 6 курсу, групи ТВ-з81мп

_____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ - 2019

Національний технічний університет України
“ Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти другий, магістерський

Спеціальність 121 “ Інженерія програмного забезпечення ”

ЗАТВЕРДЖУЮ

_____ _____
(прізвище, ініціали) (підпис)

«____» _____ 201 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри
Коваль О.В. _____
(прізвище, ініціали) (підпис)

«____» _____ 2017р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ

Пивовар Назарій Олександрович
(прізвище, ім’я, по батькові)

1. Тема дисертації Система аналізу збіжності текстової інформації для оцінки плагіату
Науковий керівник Кузьміних Валерій Олександрович, кандидат технічних наук, доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “4” листопада 2019 року № 3813-с
2. Строк подання студентом дисертації “9” грудня 2019 року
3. Об’єкт дослідження – аналіз збіжності текстової інформації
4. Предмет дослідження – система аналізу збіжності текстової інформації з використання вектору триграм
5. Перелік питань, які потрібно розробити:
- провести аналіз існуючих рішень;
- обрати бібліотеку для обробки текстової інформації і пошуку подібності;
- провести аналіз бібліотеки sklearn, яка містить утиліти для використання в обчисленні подібності текстів;
- провести аналізу веб-фреймворку Flask та бібліотеки React для побудови додатку за клієнт-серверною архітектурою;
- обрати метод для аналізу текстової інформації;
- дослідити метод векторного аналізу;
- програмно реалізувати модифікацію векторного алгоритму за допомогою триграм.

---

6. Перелік ілюстративного матеріалу 1 додаток, 39 рисунків, 16 таблиць
7. Перелік публікацій

Пивовар Н. О. Система аналізу збіжності текстової інформації для оцінки плагіату [Електронний ресурс] / Н. О. Пивовар, В. О. Кузьміних. – 2019. – Режим доступу до ресурсу: https://drive.google.com/file/d/1hJAM6oCgRw6X3il6qWjKzriu0hII-ZsG/view.

8. Дата видачі завдання «12» вересня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Постановка цілей та завдань магістерської дисертації	12 вересня 2018 р. – 10 жовтня 2018 р.	
2	Аналітичний огляд літератури, бази джерел за темою дисертації	10 жовтня 2018 р. – 14 січня 2019 р.	
3	Вивчення питань з методології та методики написання магістерської дисертації	14 січня 2019 р. – 25 січня 2019 р.	
4	Підготовка концепції дисертації	25 січня 2019 р. – 20 лютого 2019 р.	
5	Написання основних розділів автореферату	20 лютого 2019 р.. – 1 березня 2019 р.	
6	Підготовка основних розділів дисертації	26 квітня 2019 р. – 10 серпня 2019 р.	
7	Підготовка стартап-проекту	10 серпня 2019 р. – 2 вересня 2019 р.	
8	Переддипломна практика	2 вересня 2019 р. – 25 жовтня 2019 р.	
9	Захист програмного продукту	25 жовтня 2019 р.	
10	Участь у науковій конференції	14 листопада 2019 р.	
11	Попередній захист роботи	22 листопада 2019 р.	
12	Оформлення диплому	22 листопада 2019 р. – 9 грудня 2019 р.	
13	Здача всіх матеріалів на підпис зав кафедрою	9 грудня 2019 р.	
14	Захист магістерської дисертації	Грудень 2019 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Науковий керівник _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

на магістерську дисертацію

Структура й обсяг дипломної роботи. Магістерська дисертація складається зі вступу, 6 розділів, висновку, переліку посилань з 28 найменувань, 1 додатку, і містить 39 рисунків, 16 таблиці. Повний обсяг магістерської дисертації складає 95 сторінок, з яких перелік посилань займає 2 сторінки, додатки – 4 сторінки.

Актуальність теми. Розвиток та розповсюдження засобів комунікації і доступу до мережі Інтернет сприяє збільшенню спектру інформації до якої людина має доступ. За допомогою звичайного смартфону можна здійснити пошук будь-якої інформації. Тим не менш, звідси виникає нова проблема: так як кожен може отримати доступ до будь-якої інформації, значно складніше оцінити чи є вона оригіналом чи взята з інших відкритих джерел. Досить часто у мережі Інтернет можна знайти джерела інформації (будь то статті, сайти, онлайн журнали і т.п.), які майже повністю складаються с фрагментів тексту інших джерел. Сьогодні існують такі програми як Advego Plagiatius, Etxt Antiplagiat та онлайн сервіси як UNPLAG, Content-Watch та інші. Проте мало які системи дають точні результати і рекомендується перевіряти результати у декількох системах. Так як для людини це досить не проста задача було прийнято рішення розробки системи аналізу збіжності текстової інформації для оцінки плагіату.

Мета дослідження полягає у створенні системи аналізу збіжності текстової інформації для оцінки плагіату.

Завдання дослідження. Для досягнення поставленої задачі були поставлені наступні завдання:

– провести аналіз існуючих рішень;

– обрати бібліотеку для обробки текстової інформації і пошуку подібності;
– провести аналіз бібліотеки sklearn, яка містить утиліти для використання в обчисленні подібності текстів;
– провести аналізу веб-фреймворку Flask та бібліотеки React для побудови додатку за клієнт-серверною архітектурою;
– програмно реалізувати модифікацію векторного алгоритму за допомогою триграм.

Об’єктом дослідження аналіз збіжності текстової інформації.

Предметом дослідження системи аналізу збіжності текстової інформації для оцінки плагіату.

Наукова новизна одержаних результатів. Наукова новизна полягає у модифікуванні векторного методу аналізу з використанням триграм.

Практичне значення. Система аналізу збіжності текстової інформації дозволяють перевірити життєздатність розробленого алгоритму.

Ключові слова: *ЗБІЖНІСТЬ ТЕКСТОВОЇ ІНФОРМАЦІЇ, ВЕКТОРНИЙ МЕТОД, ПЛАГІАТ.*

ABSTRACT

for a master's dissertation

Structure and volume of the thesis. The master's dissertation consists of an introduction, 6 sections, conclusion, a list of links of 28 titles, 1 annex, and contains 39 pictures, 16 tables. The full volume of the master's thesis is 95 pages, of which the list of links occupies 3 pages, the applications - 4 pages.

Actuality of theme. The development and dissemination of communications and Internet access enhances the range of information that a person has access to. You can search for any information with your regular smartphone. However, this raises a new problem: since anyone can access any information, it is much more difficult to evaluate whether it is original or taken from other open sources. Quite often on the Internet you can find sources of information (be it articles, websites, online journals, etc.) that are almost entirely composed of snippets of text from other sources. Today there are programs such as Advego Plagiatius, Etxt Antiplagiat and online services such as UNPLAG, Content-Watch and more. However, few systems produce accurate results, and it is recommended that multiple systems test results. Since it is not a simple task for a person, the decision was made to develop a system of analysis of convergence of textual information for the evaluation of plagiarism.

The purpose of the study is to create text similarity analysis system for plagiarism assessment.

Objectives of the study. To achieve this task, the following tasks were set:

- analyze existing solutions;
- select a library for processing textual information and finding similarities;
- analyze the sklearn library, which contains utilities for calculating text similarities;

– analyze the Flask web framework and React library to build a client-server architecture application;
– programmatically implement the modification of the vector algorithm with the help of trigrams.

The object of the study is to analyze the text similarity methods.

The subject of the study are text similarity analysis systems.

Scientific novelty of the obtained results. The scientific novelty is to modify the vector method of analysis using trigrams.

Practical meaning. The system of analysis of convergence of textual information allow to check the viability of the developed algorithm.

Keywords: *TEXT SIMILARITY, VECTOR SIMILARITY METHOD, PLAGIARISM*.

ЗМІСТ

ВСТУП

Розвиток та розповсюдження засобів комунікації і доступу до мережі Інтернет сприяє збільшенню спектру інформації до якої людина має доступ. За допомогою звичайного смартфону можна здійснити пошук будь-якої інформації. Тим не менш, звідси виникає нова проблема: так як кожен може отримати доступ до будь-якої інформації, значно складніше оцінити чи є вона оригіналом чи взята з інших відкритих джерел. Досить часто у мережі Інтернет можна знайти джерела інформації (будь то статті, сайти, онлайн журнали і т.п.), які майже повністю складаються з фрагментів тексту інших джерел.

Сьогодні існують такі програми як Advego Plagiatius, Etxt Antiplagiat та онлайн сервіси як UNPLAG, Content-Watch та інші. Проте мало які системи дають точні результати і рекомендується перевіряти результати у декількох системах. Так як для людини це досить не проста задача було прийнято рішення розробки системи аналізу збіжності текстової інформації для оцінки плагіату.

Метою даної роботи є використання архітектури клієнт-сервер для створення програмного продукту, який представляє собою систему аналізу збіжності текстової інформації для оцінки плагіату, використовуючи модифікований алгоритм векторного аналізу.

Дана записка складається з 5 розділів. У першому розділі наведено постановку задачі короткий опис програмного продукту та задач, які він розв’язує. У другому розділі дано опис предметної області та зроблений аналіз існуючих програмних систем на базі практики. У третьому розділі представлено програмні середовища розробки, які використовувались під час написання програмного продукту. У четвертому розділі описано програмну реалізацію, та дана характеристика програмного забезпечення. У п’ятому розділі зроблено розбір роботи користувача з програмною системою, системні вимоги та запуск, а також сценарії роботи користувача з системою.

1. КОНЦЕПЦІЯ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ З ВИКОРИСТАННЯМ СТИЛЮ REST

Клієнт-серверна архітектура дуже розповсюджена для розробки більш менш великих систем. Це дозволяє розділити роботу з розроблення модулю для роботи з БД і модулю для відображення сторінки користувачеві. Також це дає можливість віддаленого доступу до системи, за допомогою мережі або веб-доступу.

1.1. Поняття клієнт-серверної архітектури

Клієнт-серверна архітектура — архітектурний шаблон програмного забезпечення, який використовують для створення розподілених мережних застосунків [12]. Застосовується для створення програмного забезпечення віддаленим зберіганням інформації та високим рівнем інтерактивності на стороні клієнта, які в подальшому використовуються для створення соціальних мереж, систем контролю виконання, систем надання колективного доступу до інформації, інтеграційних платформ, а також у науці і промисловості. Також клієнт серверну архітектуру визначають як обчислювальну модель, у якій сервер обслуговує, доставляє і керує більшістю ресурсів, для їх використання клієнтом. Такий тип архітектури має одного або більше клієнтських комп'ютерів, під'єднаних до центрального серверу через мережеве або інтернет з'єднання. Така система ділить обчислювальні ресурси.

Клієнт-серверну архітектуру також визначають як мережеву обчислювальну модель або клієнт-серверною мережею, тому що всі запити і сервіси доставляють через мережу. Така архітектура працює, коли комп'ютер клієнт посилає запит на ресурс або процес до серверу через мережеве з'єднання, яке далі обробляється і доставляється клієнту. Приклад зображено на рисунку 1.1. Сервер може

обслуговувати декілька клієнтів одночасно, як і один клієнт може бути під’єднаним до кількох серверів, кожен з яких надає різну підмножину сервісів. У найпростішій формі, інтернет також оснований на цій архітектурі, де веб-сервери підтримують роботу сайту одночасно для багатьох користувачів.

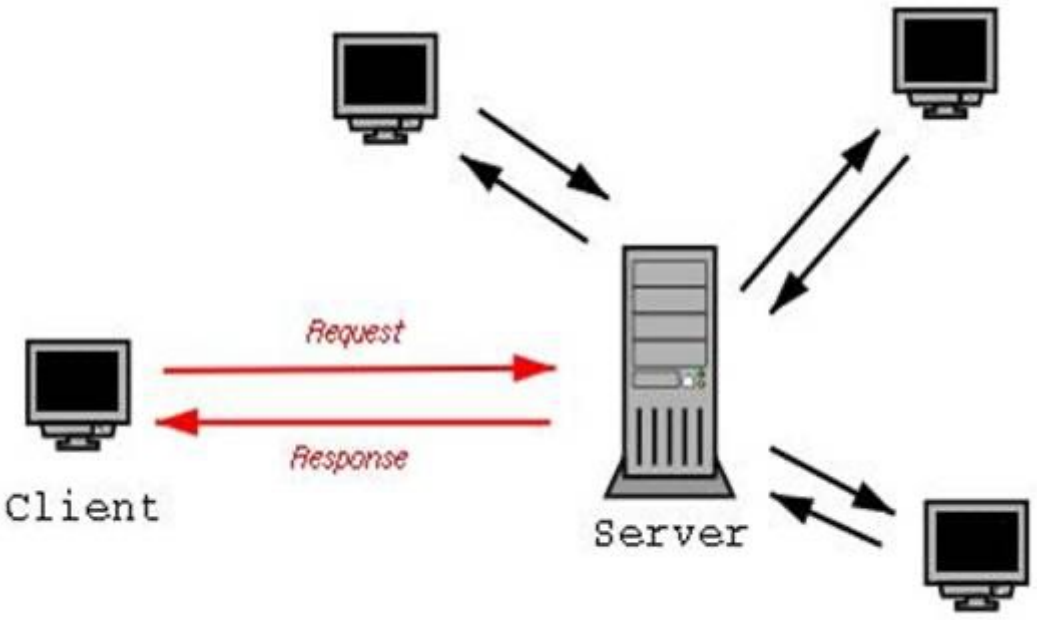

Рисунок 1.1 — Приклад роботи клієнт-серверної архітектури

Основне завдання клієнт-серверної архітектури є надання умовної цілісності розподіленим системам (сервери, клієнти і мережа) при цьому кожен компонент є досить незалежним від інших. Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного, немає жорсткого зв’язку між клієнтами і серверами. Дуже розповсюдженою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів. Так само і клієнт може звертатися то до одного сервера, то до іншого. Важливо розуміти хто або що мається на увазі коли йде мова про “клієнта”. Можна говорити про клієнтський комп’ютер, з якого відбувається звернення до інших комп’ютерів, або ж про клієнтське та

серверне програмне забезпечення. Також можна говорити про людей, які за допомогою програмних засобів мають доступ до тієї чи іншої інформації.

1.2. Основні компоненти клієнт-серверної архітектури

Завдання клієнт-серверної архітектури — забезпечити взаємодію та обмін даними між основними компонентами. При цьому компоненти можуть бути реалізовані різними програмними засобами, як показано на рисунку 1.2:

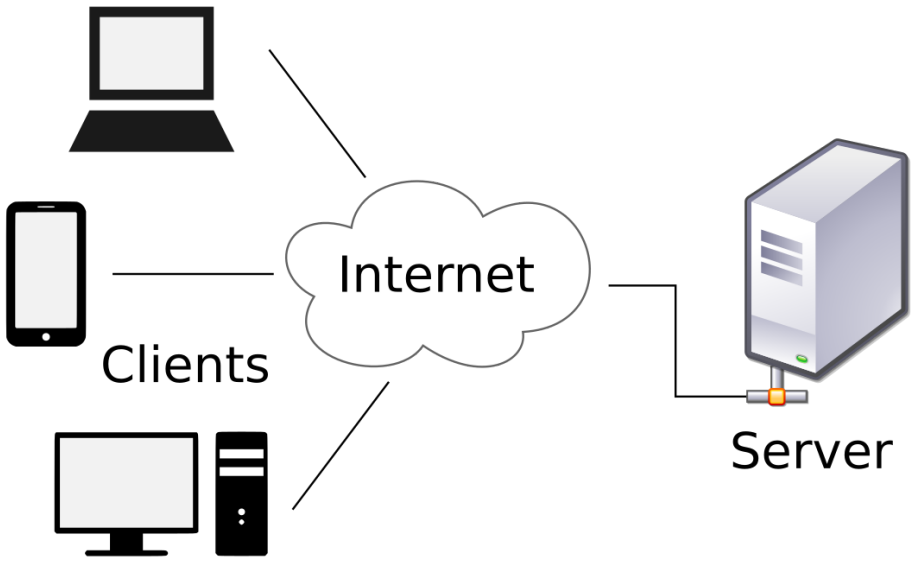

Рисунок 1.2 — Схематичне зображення сенсу клієнт-серверної архітектури

Створення клієнт-серверного програмного продукту є комплексним завданням, яке потребує багато часу та знань, а також значних вкладень у програмне забезпечення, яке коштує від кількох сотень до кількох тисяч доларів США. Також це може вимагати значних людських ресурсів, а у випадку розробки складних систем, роботи декількох фахівців або цілого відділу. Сьогодні за допомогою клієнт-серверної архітектури можна створити соціальну мережу, інтерактивний

засіб обміну повідомленнями або ж віддалене сховище даних. В цілому клієнт-серверну архітектуру можна поділити на декілька компонент:

- Сервер;
- Клієнт;
- Мережа.

1.2.1.Сервер

Сервер — надає послуги програмам що до нього звертаються. Це може бути перегляд, створення, редагування інформації і т.ін. Сервер можу бути один або бути частиною деякої групи серверів.

Сервери надають інформацію або інші можливості програмам, які до них звертаються. Функціонал, що виконуються серверами, залежить від вхідного запиту. Існує багато протоколів і методів надання функціоналу, які відрізняються між собою в залежності від того, що і як потрібно виконати. Наприклад, відображення HTML-сторінки разом зі стилями та скриптами — протокол HTTP або HTTPS.

Протокол HTTP є надійним і легким способом обміну даними у клієнт-серверній архітектурі. За основу він має розділ на методи: GET, POST, PUT, DELETE та інші. Розділ на запити є більше домовленість між розробниками, що використовують цей протокол. Майже немає принципової різниці, який метод для чого використовувати.

Часто одного методу замало, щоб вкласти всю необхідну для запиту інформацію, тому використовується спеціальна структура — заголовки запиту. Наприклад, на рисунку 1.3 зображено приклад GET запиту із заголовками, що несуть інформацію про те звідки робиться запит і яка мова тексту очікується і відповідь.

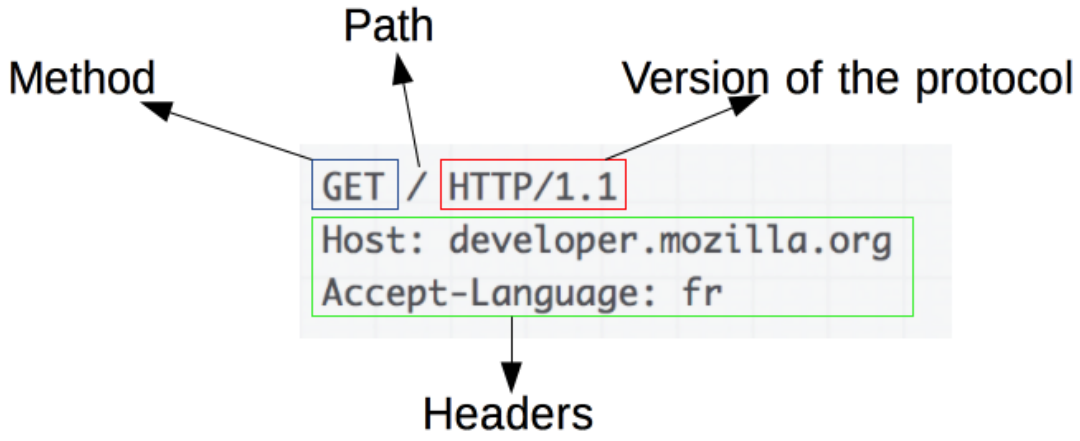

Рисунок 1.3 — Приклад GET запиту

У разі необхідності асинхронної обробки повідомлень — протокол AMQP. У його основі лежить робота з асинхронними повідомленнями. Коли клієнт отримує повідомлення він відправляє підтвердження до брокеру, що це повідомлення вже оброблено. Брокер — це сервер, який містить стан усіх повідомлень і підключених клієнтів. Він розподіляє повідомлення між вільними підключеннями клієнта. Наприклад, для протоколу AMQP брокером є RabbitMQ.

Брокер RabbitMQ надає можливість надсилання асинхронних повідомлень. Для цього він використовує черги повідомлень. Клієнт надсилає повідомлення до брокеру вказавши спеціальний routing-key ключ. Цей ключ використовується брокером для визначення до якої черги необхідно надіслати повідомлення. Клієнт же у свою чергу підписується на повідомлення якоїсь черги. Таким чином, дуже зручно налаштовується пересилка повідомлень асинхронно. Наприклад, є клієнт і сервер, що обидва використовуються протокол AMQP і брокер RabbitMQ. Коли клієнту необхідно запустити обробку якоїсь кількості повідомлень, йому немає необхідності чекати обробки цих повідомлень на сервері. Клієнт може просто відправити повідомлення на обробку і не чекати відповіді на кожне повідомлення.

Сервер зможе обробити ці повідомлення і надіслати відповідь у одразу обумовлений канал. На рисунку 1.4 зображено приклад роботи у протоколі AMQP.

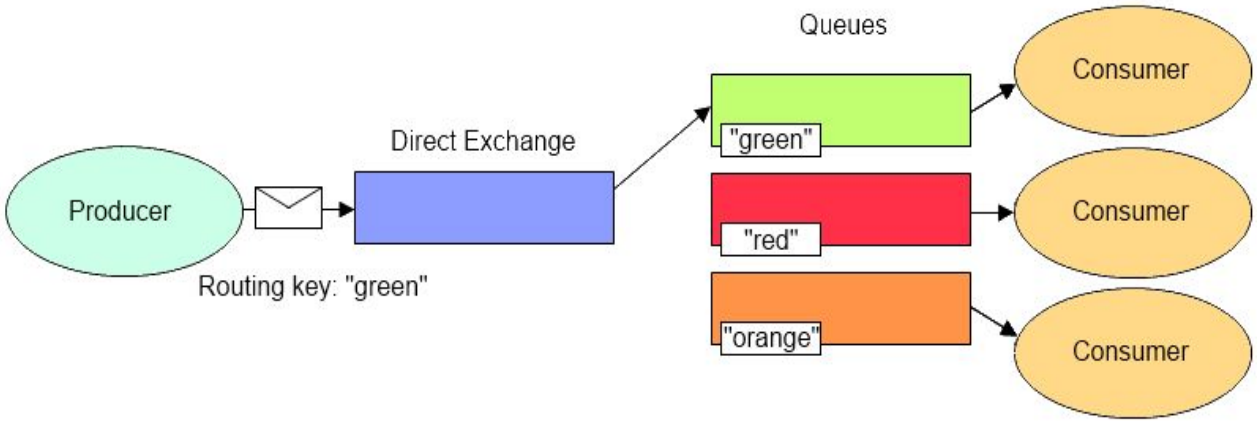

Рисунок 1.4 — Приклад відправки повідомлення у протоколі AMQP

Важливими і виділяючими факторами тут є необхідність отримання відповіді термінової відповіді від серверу, кількість одночасно виконуваних запитів, розмір повідомлення та інші. Для того, щоб сервер міг працювати з даними, він підключається до бази даних, завдання якої — зберігання структурованої інформації і швидкий пошук інформації за критерієм. За наявності програмної реалізації роботи з базою даних, сервер може і містити якусь бізнес логіку, і мати можливість працювати з даними. Сервери очікують запити від клієнтських програм і надають їм свої ресурси у вигляді даних (наприклад, завантаження файлів за допомогою HTTP, FTP, BitTorrent або робота з БД) або ж у вигляді сервісних функцій (наприклад, робота з електронною поштою, спілкування за допомогою засобів миттєвого обміну повідомленнями або просмотр веб-сторінок).

Роль сервера - це функція серверу (веб-сервер, сервер баз даних, поштовий і т.ін.). Один сервер може відігравати як одну так і декілька ролей одночасно.

Сервер, що приймає HTTP-запити від клієнтів (частіше за все веб-браузерів), видає їм HTTP-відповіді, зазвичай разом з HTML-сторінкою. Веб-сервером можна

вважати як програмне забезпечення, що виконує функції веб-сервера, так і комп'ютер, на якому це програмне забезпечення виконується. Найбільш популярними веб-серверами є Apache HTTP Server, NGINX, Apache Tomcat, Node.js, Lighttpd та інші.

Сервери баз даних використовуються для виконання користувацьких запитів до баз даних. При цьому СУБД (Система управління базою даних) знаходиться на сервері до якого підключаються клієнтські застосунки.

Файловий сервер - зберігає інформацію у файловій системі та надає користувачам доступ до неї.

Сервер друку - надає можливість віддаленого доступу до принтерів. Дозволяє друкувати через URL принтера, використовуючи протокол IPP, а також підключати принтери, використовуючи Point to Point.

Поштовий сервер - обслуговує основні поштові адреси користувачів системи і дозволяє приймати і відправляти пошту. Схема роботи поштового серверу зображено на рисунку 1.5:

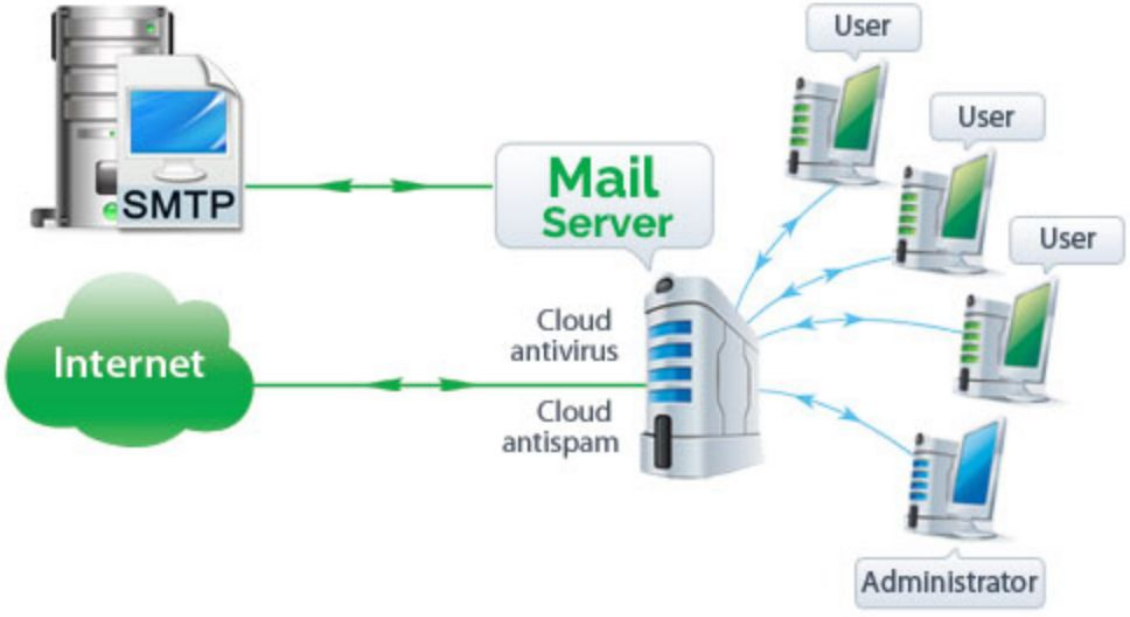

Рисунок 1.5 — Приклад роботи поштового серверу

VPN сервер - надає точку входу в мережу системи для віддалених користувачів. Використовується для обмеження доступу до внутрішніх компонентів системи. Дозволяє реалізовувати протоколи маршрутизації для середовищ LAN, WAN. Підтримує модемні з'єднання і VPN через інтернет.

DNS сервер - дозволяє перетворювати доменні імена (FQDN) в адреси IP. Без цього у мережі інтернет не може працювати жоден пристрій. Коли у браузері вводиться адреса сайту. Для отримання IP адреси, з якою можна буде працювати, пристрій надсилає запит до DNS-серверу. Сервер відповідає клієнту IP адресу, яка потім використовується пристроєм для роботи з цим сайтом Приклад роботи DNS серверу зображено на рисунку 1.6:

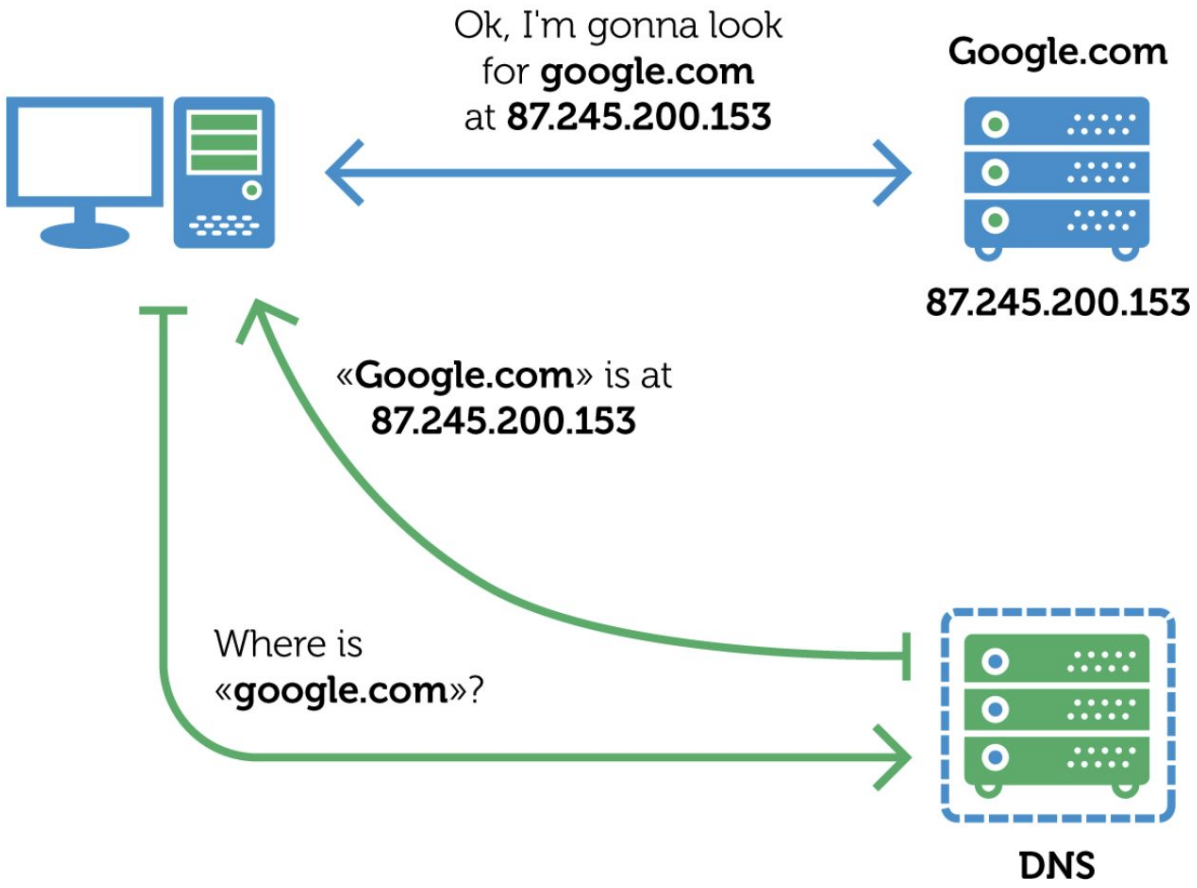

Рисунок 1.6 — Приклад роботи DNS серверу

DHCP сервер - дозволяє клієнтам отримувати свій IP за потребою. Сервер DHCP також надає додаткову інформацію для конфігурації мережі — адреса серверів DNS, WINS і т.ін.

1.2.2. Клієнт

Дуже важливо розуміти, що мається на увазі, коли мова йдеться про “клієнта”. Можна говорити про клієнтський ноутбук, з якого відбувається звернення до інших комп’ютерів. Можна говорити про клієнтське та серверне програмне забезпечення. Нарешті, можна говорити про людей, які за допомогою програмного забезпечення мають доступ до тієї чи іншої інформації. Прийнято вважати, що клієнти та сервери - це перш за все програмні модулі. Зазвичай вони знаходяться на різних комп’ютерах, але бувають ситуації, коли обидві програми - і клієнтська, і серверна, фізично розміщуються на одній машині. В такій ситуації сервер часто називається локальним. В клієнт-серверній архітектурі клієнт - це рівень представлення даних, який являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього команд. У випадку дворівневої клієнт-серверної архітектури має місце взаємодія двох програмних модулів - клієнтського та серверного. В залежності від того, як між ними розподіляються обов’язки, розрізняють модель тонкого та товстого клієнта.

Тонкий клієнт - програма, що забезпечує тільки функції рівня представлення. В той час як вся логіка застосунку зосереджена та управління даними зосереджені на сервері.

Товстий клієнт - модель, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами також називають пристрої з обмеженою потужністю: кишенькові комп’ютери, мобільні телефони і т.ін.

1.2.3. Мережа

Мережа - сукупність комп’ютерів, об’єднаних за допомогою каналів зв’язку та засобів комутації в єдину систему для обміну повідомленнями і доступу користувачів до програмних, технічних, інформаційних і організаційних ресурсів мережі.

Комп’ютерна мережа представляє собою сукупність вузлів (комп’ютерів і мережевого обладнання) і об’єднучих гілок (каналів зв’язку). Схематично це зображено на рисунку 1.7:

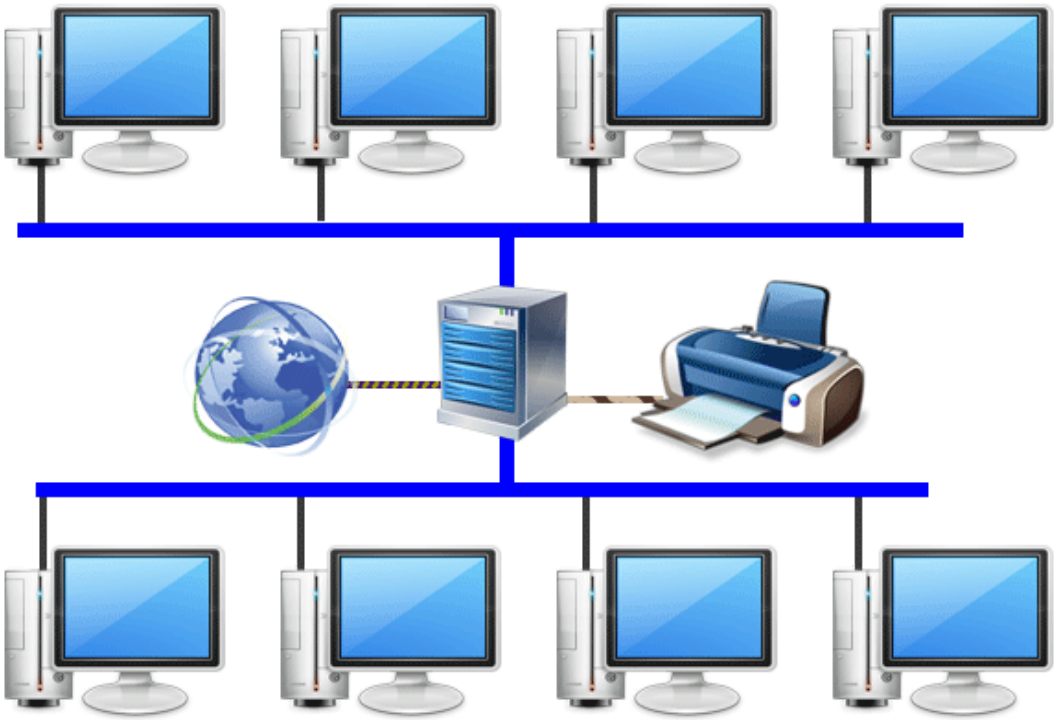

Рисунок 1.7 — Схематичне зображення комп’ютерної мережі

Гілка мережі - це шлях, об’єднуючий два зміжних вузли. Розрізняють кінцеві вузли, розташовані у кінці тільки однієї гілки, проміжні, розташовані на кінцях більше ніж однієї гілки та суміжні - такі вузли об’єднані щонайменше одним

шляхом, що не містить ніяких інших вузлів. Комп'ютери можуть бути об'єднані в мережу різними способами.

Логічний і фізичний способи з'єднання комп'ютерів, кабелів та інших компонентів, що містяться у мережі, називаються топологією. Топологія характеризує властивості мережей, що не залежать від їх розміру. При цьому враховується потужність та принцип роботи цих об'єктів, їх типи, довжини каналів, адже при проектуванні ці фактори дуже важливі.

1.3. Методологія REST

Методологія REST створена для побудови розподілених застосунків. Ключовою абстракцією є ресурс. Будь-яка інформація може бути ресурсом (документ, зображення, колекція інших ресурсів і т.ін.).

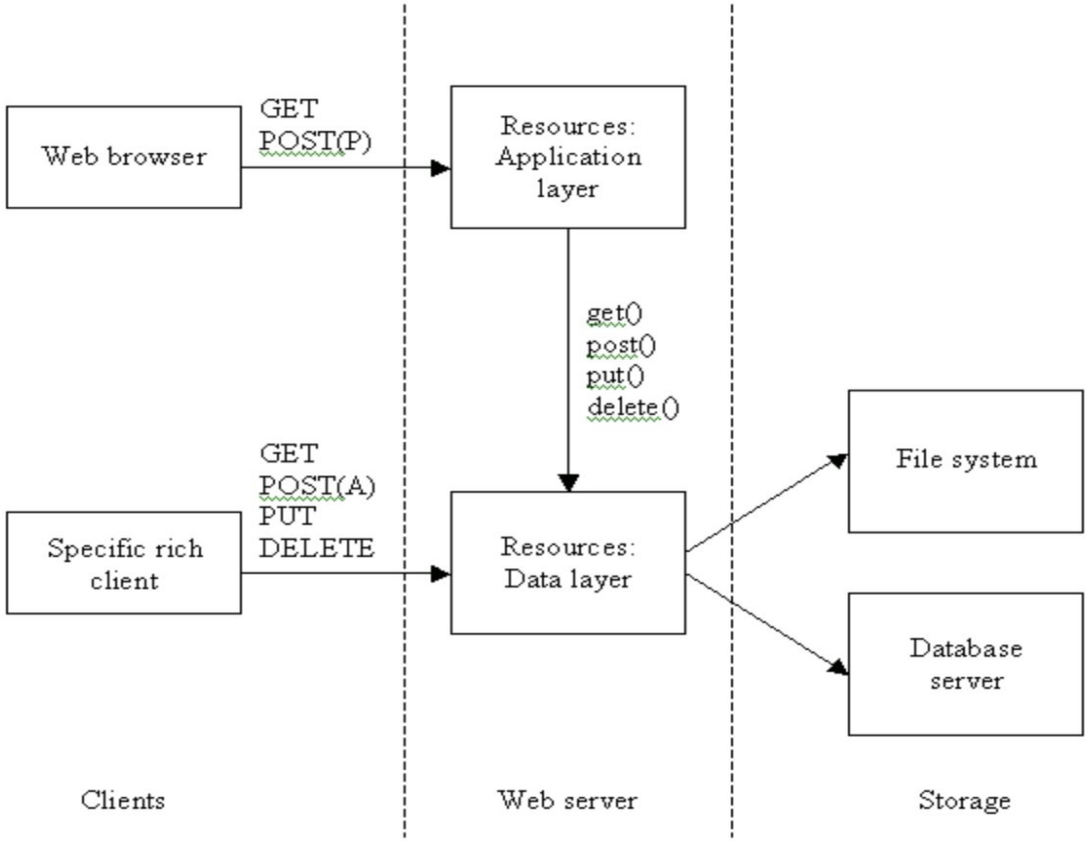

Рисунок 1.8 — Принцип роботи системи за REST архітектурою

Ресурс має стан, і може бути отриманий чи змінений за допомогою представлень. Сукупність станів ресурсів і є станом застосунку.

Представлення - це дані (JSON, XML, HTML або текст) у спеціальному форматі, що дозволяє однозначно визначати стан ресурса. Застосунок відповідає за деяку множину таких ресурсів. При використанні JSON для кодування інформації, тіло звичайного запиту може виглядати як на рисунку 1.9:

Рисунок 1.9 — Приклад JSON закодованої інформації

Представлення, яке модифікує стан ресурса, і представлення, яке дозволяє отримати інформацію про стан ресурса, не обов’язково повинні відповідати один одному. Концепція Hypermedia as the Engine of Application State (HATEOAS) дає можливість клієнту розуміти у який стан він може перевести ресурс, і як він може отримати ці стани. В даному випадку дії, які можна проводити з ресурсом, обраховує сервер. Клієнту не потрібно володіти логікою для обрахунку можливих операцій і йому немає необхідності знати адреси операцій. Найголовніше тут, що клієнт знає тільки одну точку входу, тільки один URL. А далі він отримує представлення разом зі списком можливих дій і може приймати відповідні рішення. Це на порядки полегшує спростити клієнт, так як йому не потрібно буде зберігати логіки, а лише покладатися на адреси.

Одна із складних тем в методології REST - моделювання ресурсів. Тут не існує єдиного підходу або простого правила, яке дозволить точно підібрати межі ресурсу. Необхідно проектувати API так, щоб воно не могло вивести застосунок із ладу. Найпопулярнішим, наприклад, є підхід CRUD (Create Read Update Delete). Його схематично зображено на рисунку 1.10. Підхід полягає в тому, що для читання ресурсів використовуються GET запити, для створення POST запити, для оновлення ресурсів PATCH або PUT і для видалення ресурсів використовується DELETE. Це дозволяє досить логічно розділити запити до сервера за типом дії.

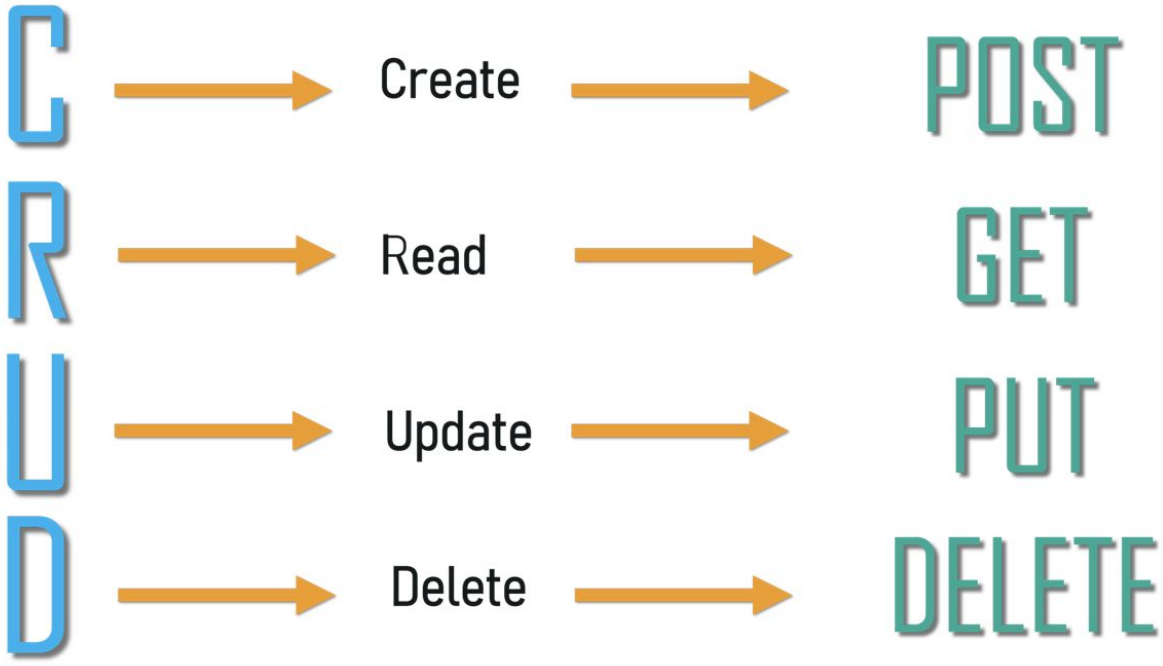

Рисунок 1.10 — Схема підходу CRUD

Існує достатня кількість бібліотек для побудови інтерфейсу користувача у браузері, які базуються на тому, що сервер працює за підходом CRUD. Це значно прискорює розробку інтерфейсу і надає розробнику можливість зосередитись лише на комфорті користувача.

У більшості випадків можна просто використати встроєне API для виконання HTTP запитів. Наприклад, на даний момент у браузері є JavaScript функція fetch. На

рисунку 1.11 зображено приклад виконання POST запиту з використання цієї функції.

```
async function postJson (args = {}) {
    const { body, url, token } = args;
    assert.ok(body, 'Body is required');
    assert.ok(url, 'Url is required');
    const r = await fetch(url, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Authorization': 'Bearer ' + token
        },
        body: JSON.stringify(body)
    });
    if (errorsMap[r.status]) {
        throw errorsMap[r.status](await r.text());
    }
    return await r.json();
}
```

Рисунок 1.11 — Приклад використання fetch API

Деякі бібліотеки підняли комфорт розробника до нового рівня. Звичайну REST CRUD архітектуру можна використати з новим підходом для організації запитів, який називається JSONAPI. Цей підхід полягає в тому щоб тіло кожного запиту і відповідь до кожного запиту була стандартизована. Тобто щоб те що приймає POST запит завжди у такому ж форматі поверталося на GET запит. Це досягається за рахунок того, що інформація про створення ресурсу завжди лежить у полі data. Інші поля використовуються для метадати. Наприклад, для того щоб вказати прив’язку до іншого ресурсу.

Дуже цікавим є і отримання ресурсів за такого підходу. Запит типу GET містить складну структуру. Поле data містить масив об’єктів і кожен з них має містити посилання на отримання інформації саме по цьому елементу масиву. На рисунку 1.12 зображено приклад відповіді серверу, який програмно реалізовує підхід JSONAPI.

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "links": {
    "self": "http://example.com/articles"
  },
  "data": [{
    "type": "articles",
    "id": "1",
    "attributes": {
      "title": "JSON:API paints my bikeshed!"
    }
  }, {
    "type": "articles",
    "id": "2",
    "attributes": {
      "title": "Rails is Omakase"
    }
  }]
}
```

Рисунок 1.12 — Приклад відповіді JSONAPI GET запиту

Зважаючи на вищесказане, можна зробити висновок, що JSONAPI складний, але дуже цікавий підхід реалізаціїї CRUD системи. Набагато більше користі цей підхід приносить для розробників користувацького інтерфейсу на JavaScript, а саме

гарна підтримка у бібліотеках. Наприклад, фреймворк Ember.js надає розробнику спеціальний JsonApiAdapter, який надає можливість взагалі не писати ніякого коду для здійснення запитів до серверу, якщо сервер реалізує доступ до ресурсів за підходом CRUD. Таким чином, розробник інтерфейсу у себе в коді просто робить виклик функції з аргументом типу ресурсу, який йому потрібен і все.

Не існує якогось єдиного стандарту, який би повністю описував REST. Але існує багато невеликих стандартів, які доповнюють стиль:

- URI і URI Template - про унікальні ідентифікатори ресурсів;
- HTTP - як протокол передачі текстової інформації;
- HAL, Siren - для реалізації HATEOAS;
- JSON, XML - для представлень;
- HTML - можна використовувати не тільки для представлень, але й запозичувати ідеї для компонент гіпермедіа.

У REST чудово те, що цей архітектурний стиль чудово проектується на HTTP, тобто будь-яка бібліотека, що працює з HTTP, підійде. І для HATEOAS індустрія не стоїть на одному місці. Існують бібліотеки, що значно полегшують роботу з побудовою адрес. Для програмування на Java слід звернути увагу на Spring HATEOAS, для PHP - Hateoas.

У наш час можна знайти все більше і більше прикладів використання принципів REST. Наприклад, Github API дуже широко застосовує гіперпосилання. Взагалі, кожна більш-менш навантажена система переходить на цю архітектуру, адже це надає можливість незалежно збільшувати потужність на сервері і економити на ресурсах інтерфейсу. Потім же зекономлені ресурси, можна або направити на посилення серверів, або ж зберегти гроші і вкласти їх у подальший розвиток проекту.

Висновок до розділу 1

Розробка системи, що має клієнт-серверну архітектуру — один із найпопулярніших напрямів реалізації розподілених систем, причому створюються вони не лише для специфічних цілей, а й для дуже поширених задача, як наприклад створення інтернет-магазину. Можна зробити висновок, що клієнт-серверна архітектура стала невід'ємною частиною розподілених систем, сфери її застосування постійно збільшуються, що, в свою чергу, вимагає більш детального вивчення питань, пов'язаних з забезпеченням якісної взаємодії між всіма компонентами розподілених систем.

Одним з основним недоліків під час роботи з традиційними розподіленими системами є те, що на побудову взаємовідносин витрачається велика кількість часу. Системи, які розробляються, за допомогою методології REST, дозволяють керувати всіма ресурсами системи через виділене API.

2. АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ПРИ РОЗВ'ЯЗАННІ ЗАДАЧІ РОЗРОБКИ СИСТЕМИ АНАЛІЗУ ТЕКСТОВОЇ ІНФОРМАЦІЇ ДЛЯ ОЦІНКИ ПЛАГІАТУ

У наш час існує велика кількість систем для аналізу збіжності текстової інформації для оцінки плагіату. Вони можуть бути більш направленими на наукові статті, доповіді до конференцій або ж студентські роботи, але принцип дії у них усіх схожі.

2.1. Поняття плагіату

Плагіат — привласнення авторства на чужий твір або на чуже відкриття, винахід чи раціоналізаторську пропозицію, а також використання у своїх працях чужого твору без посилання на автора [24]. Таким чином, плагіатом можна вважати: викрадення чужої ідея або видання її за свою; використання результатів чужої роботи без вказання джерела.

Плагіат є небезпечним тому, що людина може втратити будь-яку мотивацію для створення нових статей, робіт і т.п., якщо не буде ніякої гарантії, що її робота не буде вкрадена і використана під чужим іменем. Для того ж щоб визначити плагіат, потрібно володіти інформацією про всі опубліковані роботи і власне самою темою.

Саме якісний контроль оцінки робіт на плагіат може захистити авторське право автору оригіналу.

2.2. Методи аналізу збіжності текстової інформації

Наразі існує багато різних методів аналізу збіжності. Всі вони мають спільну реалізацію: представлення текстів як вектори фактів, і потім обрахування відстані між цими фактами. Основні методи: збіжність Джакарда, К-середніх, косинус подібності, векторний метод та стохастичний. Реалізація, наприклад, методу Джакарда [26] полягає у відношенні довжини множини спільних фактів до довжини всіх фактів. Основна ідея показана на рисунку 2.1.

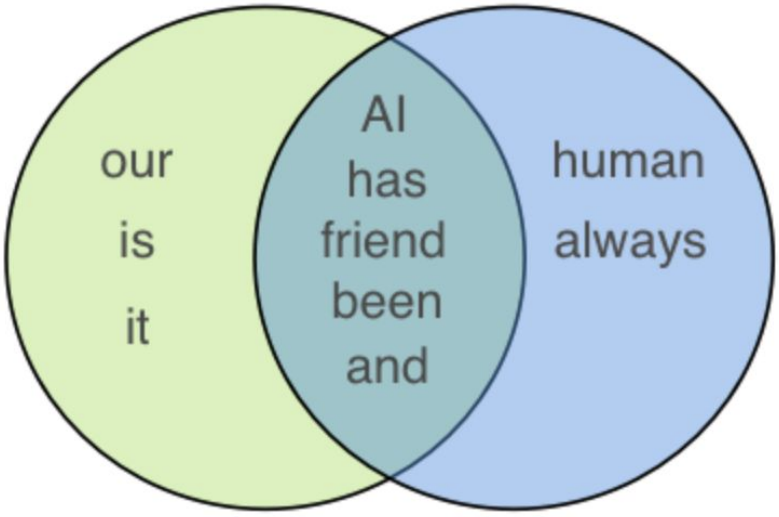

Рисунок 2.1 — Приклад методу збіжності Джакарда

Цей метод дозволяє отримати подібність двох текстів. Правду кажучи, його корисність не обмежується використанням у аналізі подібності двох текстів. Його можна використовувати для оцінки подібності будь-яких об’єктів за будь-якими критеріями. Наприклад, якщо треба оцінити подібність двох користувачів соціальної мережі. У даному теоретичному випадку ми можемо дізнатися інформацію про користувача у базі даних. Маючи інформацію про те, до яких спільнот належить кожен користувач у даній системі, можна зробити висновок про подібність інтересів цих користувачів. Тобто, якщо користувач 1 належить до спільнот А, Б і В, а

користувач 2 — до Б, В і Г, то можна зробити висновок що їх інтереси є досить подібними за рахунок спільних спільнот Б і В.

Зважаючи на користь, яку цей алгоритм приносить і простоту виконання, його можна досить успішно використовувати для нескладних аналітичних задач кластеризації користувачів по інтересам, задач по проблема, тощо.

Також дуже вагомим фактором на користь даного алгоритму є його програмна реалізація у більшості аналітичних бібліотек. Наприклад, у Python бібліотеці для роботи з нейронними мережами sklearn. Можна просто завантажити необхідну функцію у свій модуль і використовувати її. На рисунку 2.2 наведено приклад використання методу Джакарду у бібліотеці sklearn:

Examples

```
>>> import numpy as np
>>> from sklearn.metrics import jaccard_similarity_score
>>> y_pred = [0, 2, 1, 3]
>>> y_true = [0, 1, 2, 3]
>>> jaccard_similarity_score(y_true, y_pred)
0.5
>>> jaccard_similarity_score(y_true, y_pred, normalize=False)
2
```

Рисунок 2.2 — Приклад використання методу збіжності Джакарда

Такі переваги, зазвичай, є визначальними у популяризації тієї чи іншої технології, бо вони полегшують порог входу до використання даного алгоритму. Наприклад, якщо у розробника є проблема і він дізнається, що він може її вирішити використовуючи алгоритм А і, що він має програмну реалізацію у бібліотеці Б. За такими обставинами, розробнику немає необхідності досконало розбиратися у нюансах роботи даного алгоритму. Він може просто почати використовувати цей

алгоритм і вирішувати свою проблему. Зважаючи на простоту використання, він буде рекомендувати цей алгоритм колегам для вирішення їх проблем. Коли ж йому стане зрозуміло, що з якоїсь причини алгоритм перестав відповідати його задачам, він зможе розібратися у деталях алгоритму і вирішити, що саме для нього не є оптимальним. Або ж розробник зможе знайти модифікацію даного алгоритму, яка є більш оптимізованою під його задачі, або ж просто більш швидкою. Тобто головним у програмуванні, є те скільки часу потрібно витратити на вирішення однакових проблем різними способами, і популярними будуть ті інструменти, які краще підтримують баланс між прогресивність алгоритму і простотою використання у різних середовищах.

Мінусом цього методу являється те, що звичайна заміна слів на синоніми значно впливає на результати. Тобто це означає, що якщо системі дати декілька текстів, написаних на різні теми, але з однієї галузі, то велика вірогідність, що вони будуть мати спільні слова. Це є мінусом, тому що можна отримати хибний результат аналізу текстів, проте для системи це буде абсолютно правильним результатом.

Слово не може оцінюватися як запатентована оригінальна думка, тому що слово належить мові. Ніхто не може заборонити вам використовувати якесь літературне слово. Менше дробити фактори оцінки немає сенсу, бо ми отримаємо таку ж ситуацію із ще більшою похибкою.

2.3. Векторний метод представлення текстової інформації

Одним із найзручніших способів представлення текстової інформації є векторний метод. Він обігрує багато інших методів тим, що по суті своїй є дуже швидким і тим, що залишає простір для вдосконалення. Швидкість методу аналізу текстової інформації є дуже важливою складовою системи аналізу збіжності текстової інформації, адже від цього буде напряму залежати привабливість продукту для споживача. Якщо продукт буде точним, але неймовірно повільним, то мало хто

матиме змогу витрачати стільки часу на перевірку оригінальності текстів. У деяких випадках цього часу може просто не вистачати для уникнення якихось бюрократичних умов. От якщо користувач, може сам виставити не надто високу точність хоча б для перевірки правильності роботи системи, то це значно збільшить можливість того, що він стане постійним користувачем даної системи. У наш час, коли все змінюється із неймовірною швидкістю і новина, яка актуальна і важлива зараз, може стати нікому не важливою завтра. Наприклад, якщо необхідно терміново перевірити звідки була взята інформація у відкритому джерелі, може просто не бути достатньо часу, щоб виконати перевірку у срок, у який дана стаття буде взагалі мати якийсь сенс. Можливо у такому випадку простіше і дешевше буде просто почекати поки стаття перестане бути актуальною, аніж займатися перевіркою її оригінальності. На рисунку 2.3 зображено приклад використання модулю CountVectorizer:

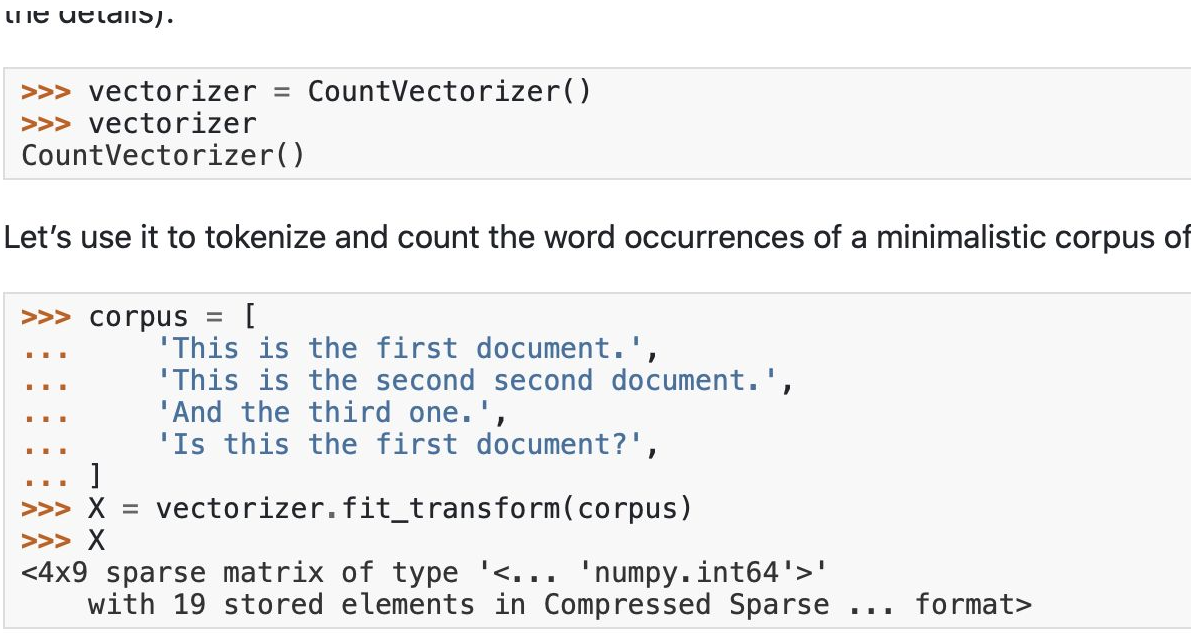

Рисунок 2.3 — Приклад використання модулю CountVectorizer

Суть методу полягає у тому, в одній із реалізацій, розбити текст на слова і представити це як послідовність токенів. У такому вигляді з отриманих токенів

видаляються дублікати і обраховується скільки разів кожне слово (токен) зустрічається у тексті. Після всіх цих маніпуляцій ми маємо вектор тексту, який мається готову для обробки інформацію про те, які слова у в якій кількості присутні у тексті.

Для розуміння алгоритму необхідно більш детально розібратися у всіх етапах обробки тексту. Етапів не дуже багато, але вони представлені такими: етап створення словника, етап визначення розміру і типу токену, етап розбиття тексту на токени, етап пошуку кількості токенів у тексті і етап запису вагів відповідних токенів у результуючий вектор.

Етап створення словнику є одним із найважливіших. Він полягає у тому що відібрати всі символи і знаки у відповідну структуру, звідки їх можна буде передавати далі. На даному етапі, відфільтровуються пунктуаційні знаки, цифри і таке інше. Це необхідно для зменшення кількості інформації для обрахування. По-друге це необхідно для того щоб не використовувати пунктуаційні символи для оцінки подібності двох фрагментів текстової інформації, бо пунктуаційні символи не належать жодній людини, вони використовуються мовою для побудови тих чи інших конструкцій. Так само не можна для оцінки подібності двох текстів використовувати загальноприйняті формули. Якщо ж людина запатентує формулу води і заборонить використання цієї формули. Як у таких умовах вести якусь наукову діяльність, коли навіть не можна використовувати воду у своїх роботах. А вода це ж складова майже будь-якої рідини. Або ж якщо взяти формулу вільного падіння, то як у такому випадку розробляти якісь експерименти, які пов’язані із предметами, які падають з якоїсь висоти. Або ж як проводити тести машин на стійкість до аварій, бо тут же використовуються закони Ньютона. Саме тому етап створення словнику є таким важливим.

Наступний етап — це етап визначення розміру і типу токену. Текст під час обробки розділяється на так звані “токени”, які використовуються у подальшій обробці. Це дозволяє стандартизувати вигляд усіх вихідних векторів. Тобто якщо

аргументами функції є багато текстів, які необхідно перевірити на подібність, то необхідно виділити повний список всіх токенів, які належать хоча б одному тексту. Для цього створюється список токенів і у циклі відбувається проходження усім текстами і додання нових токенів у список. Наприклад, якщо ми візьмемо 2 тексти які містять різні слова і спільна слова, то до списку токенів будуть додані слова, які належать тільки першому тексту, слова які належать тільки другому тексту і слова які є спільними для обох текстів. Це дозволить оцінити вагу кожного токену у тексті на подальших етапах. Додання ж токенів, які містяться хоча б в одному тексті дає можливість оцінити ситуацію, коли у якихось текстах слово зовсім не представлено. У такому випадку вага такого токену будо 0.

На етапі пошуку кількості токенів у тексті ми уже маємо список токенів, які необхідно шукати. Ця оптимізація відіграє дуже важливу роль. Так як не потрібно виконувати ніяких додаткових обчислень, то це досить вагомо збільшує швидкість виконання алгоритму. Саме для цього і необхідно розбиття на етапи. Все це дає можливість заново використовувати результати виконання попередніх етапів у майбутніх етапах. На даному етапі, використовуючи список токенів і обраховується кількість кожного токену для кожного тексту. Це число записується у відповідну вагу. Пошук входжень токену у текст швидше за все може виконуватись, за допомоогою регулярних виразів, що присутні майже у всіх більш-менш розвинутих мовах програмування. Адже це дозволяє задати правило по якому шукати і просто знайти кількість входжень цього токену у тексті. А вже програмна реалізація мови відповідає за швидкість виконання регулярного виразу.

На етапі створення векторів використовуються обчислені ваги токенів для кожного тексту. Знову ж таки це дуже гарна оптимізація виконання, що значно прискорює швидкість виконання алгоритму. Зберігаючи порядковий номер кожного токену, його вага записується у відповідний елемент масиву. Коли всі ваги для тексту записані у масив, то цей масив можна вважати вектором цього тексту у N-просторі, де N — кількість токенів.

Успішне виконання всіх цих етапів, забезпечує системі швидке виконання операції формування векторів. Також сам факт того, що використовується модуль, що вже надає даний функціонал, його не потрібно заново програмно реалізовувати. Це економить час, який можна використати на подальше вдосконалення системи і алгоритму. Також, так як модуль sklearn є досить популярним, реалізація даного алгоритму вже є відлагодженою і оптимізованою. Те що багато людей використовують цей алгоритм, і надсилаються свої рекомендації авторам модулю, дає можливість розраховувати на коректність і оптимальність використаного у системі алгоритму.

2.4. Метод збіжності векторів за косинусом подібності

Отримані вектори необхідно якимось чином обробляти, щоб отримати результат аналізу. Одним з методів обробки текстових векторів є метод косинусу подібності. Сама формула не є складною і ми її добре пам’ятаємо з шкільного курсу тригонометрії:

$$cos(\theta) = \frac{A \times B}{|A||B|} \tag{2.1}$$

де A та B — вектори.

За допомогою цієї формули можна було обчислити косинус кута між двома векторами, а значить і сам кут. Водночас, її можна застосувати і до аналізу збіжності текстів, тому що формула оперує з векторами, і ми можемо привести текстову інформацію до вигляду векторів. Звідси, підставивши у формулу наші вектори, ми можемо отримати певну оцінку їх збіжності.

Для зменшення час обчислення цих коефіцієнтів, використовують матриці. Спочатку формується матриця векторів, як показано на рисунку 2.4:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Рисунок 2.4 — Матриця векторів

У даній матриці кожен рядок — це частота вживання певного слова у тексті. Кожен стовпчик матриці — це вектор. Далі, необхідно нормалізувати значення матриці, тобто привести значення до проміжку від 0 до 1. Отже, звідси маємо наступну нормалізовану матрицю, як показано на рисунку 2.5:

$$\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix}$$

Рисунок 2.5 — Матриця з нормалізованими значеннями

Для векторного перемноження матриці векторів, нам необхідно спочатку отримати транспоновану версію нормалізованої матриці, таку матрицю зображено на рисунку 2.6. Для того щоб отримати транспоновану версію матриці, якщо простими словами, то необхідно змінити стовпці і рядки матриці місцями. Тобто якщо стовпець розташований у матриці вертикально — у матриці він буде замість відповідного рядка, а рядок — на місці стовпчика.

$$\begin{bmatrix} b_{11} & b_{21} & \dots & b_{m1} \\ b_{12} & b_{22} & \dots & b_{m2} \\ \dots & \dots & \dots & \dots \\ b_{1n} & b_{2n} & \dots & b_{mn} \end{bmatrix}$$

Рисунок 2.6 — Транспонована матриця

Отримавши транспоновану матрицю ми можемо виконати векторне перемноження матриці векторів. У результаті ми отримаємо матрицю, яку зображено на рисунку 2.7:

$$\begin{bmatrix} W_{11} & W_{12} & \dots & W_{1n} \\ W_{21} & W_{22} & \dots & W_{2n} \\ \dots & \dots & \dots & \dots \\ W_{n1} & W_{n2} & \dots & W_{nn} \end{bmatrix}$$

Рисунок 2.7 — Матриця коефіцієнтів подібності кожного вектора до кожного іншого вектора

Ця матриця надає інформацію щодо коефіцієнту подібності кожного вектора відносно кожного іншого вектору. Наприклад, у звичайному випадку перший коефіцієнт першого рядку буде 1, тому що цей коефіцієнт це відношення даного вектора до нього самого. Якщо ми хочемо отримати інформацію щодо відношення подібності інших векторів до першого, то необхідно взяти перший рядок цієї матриці і взяти коефіцієнти починаючи з другого. Таким чином, ми отримаємо інформацію щодо подібності текстів відносно першого тексту.

Робота з матрицями, а не з окремими векторами дає нам можливість набагато швидше обчислювати необхідну нам інформацію. Адже для цього нам достатньо

використати інструменти оптимізовані для роботи з матрицями, такі як Numpy. За допомогою цієї бібліотеки ми можемо без особливих зусиль використовувати матричні операції у Python середовищі, просто використовуючи двовимірні масиви чисел.

2.5. Модифікація векторного методу

Векторний метод відносно добре працює з простими текстами, але ми можемо отримати похибку на текстах з великою кількістю однакових слів з різним сенсом. Та сама похибка буде, якщо слова будуть омонімами. Також, якщо модифікувати текст, замінити деякі слова на синоніми і т.п., векторний метод втрачає свою ефективність. Тобто на текстах із великою кількістю однакових слів, алгоритм почне давати високий коефіцієнт подібності, хоча це буде зовсім не вірно. У таких випадках система повинна якимось чином визначати, що слова використовуються із різним сенсом. Таке формулювання задачі може означати використання штучного інтелекту для розуміння контексту використання слова, для того, щоб розуміти у якому сенсі те чи інше слово використано у тексті. Проте це було б надто важким шляхом вирішення даної проблеми. Системи штучного інтелекту ще не досягли достатньо високого рівню розвитку для того щоб вирішувати подібні проблеми, або ж розробка таких систем є надто складною для даного проекту. Саме тому було вирішено продовжити пошуки модифікації векторного алгоритму для того, щоб система залишалася простою і зберігала настільки мало стану наскільки можна.

Саме для вирішення таких проблем було застосовано модифікацію векторного методу, що використовує триграми слів. Триграми у рамках даного алгоритму означають групування токенів на рівні слів у групи по 3.

Для цього на етапі формування списку токенів відбувається дописування додаткових токенів, які є комбінаціями конкретного слова з його сусідами зліва та

справа. Слід відзначити, що це збільшує кількість операцій, які необхідно виконати для обчислення векторів. Але це робить метод більш чутливим до контексту тексту. Основною перевагою даного підходу, що контекст тут формується із комбінації слів, а не за допомогою якоїсь складної системи штучного інтелекту. Таке рішення робить алгоритм не таким складним, яким він міг би бути. Це дуже важливо зберігати алгоритм настільки простим, наскільки це можливо, адже це дозволяє модифікувати та вдосконалювати алгоритм із меншими часовими та людськими затратами.

Висновок до розділу 2

У наш час існує велика кількість методів аналізу збіжності текстової інформації для оцінки плагіату, але принцип дії у них усіх схожі.

Є різні метод для перетворення тексті до внутрішнього представлення, біль зручного для обробки алгоритмами, та різні алгоритми обробки цих самих представлень.

У даній роботі використовується модифікований векторний метод перетворення текстів до внутрішнього представлення, а порівнюються ці вектори методом збіжності косинусу подібності.

3. ЗАСОБИ РОЗРОБКИ

Під час розробки програмного продукту використовувалися веб-фреймворк Flask, бібліотеки React та jQuery, інструмент для збірки JavaScript файлів Webpack, програмну платформу Node.js, середовище розробки Visual Studio Code, інструмент для візуального проектування БД MongoDB Robo3t, сам сервер БД MongoDB та інструменти розробки Google Chrome DevTools.

3.1. Огляд веб-фреймворка Flask

Мікро-фреймворк Flask - являє собою мінімалістичний інструмент для створення веб-додатків, REST серверів та інше. Мікро не означає, що весь програмний код має бути у одному Python файлі, або ж що Flask-у не вистачає функціоналу. Це означає що ядро фреймворка просте, але легко розширюється. Фреймворк не приймає рішень щодо БД, яку використати тощо. Все це може визначити користувач. За замовчуванням, Flask не включає жодного функціоналу з роботи з БД, або перевірки вхідних даних. Замість цього фреймворк підтримує розширення, які надають додатку необхідний функціонал. Розширення можуть бути різними: робота з БД, перевірка вхідних даних, функціонал завантаження, різні методи аутентифікації. Фреймворк хоч і називається “мікро”, але він точно готовий до використання у нормальних робочих умовах.

Як тільки ваш Flask-додаток створено, ви зможете знайти різноманітні розширення доступні у спільноті, які головна команда розробки Flask ретельно перевіряє. Зі зростанням кодової бази, Flask продовжить надавати зручний спосіб для вас контролювати розробку та розвиток вашого додатку.

Архітектура Flask додатків дуже сприяє тому, що розробник використовує тільки ті модулі, які необхідні йому для вирішення даної задачі. На рисунку 3.1

зображено обсяг коду який необхідно ввести, щоб розробити перший найпростіший веб-сервер на Flask:

```
from flask import Flask, escape, request

app = Flask(__name__)

@app.route('/')
def hello():
    name = request.args.get("name", "World")
    return f'Hello, {escape(name)}!'
```

```
$ env FLASK_APP=hello.py flask run
 * Serving Flask app "hello"
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Рисунок 3.1 — Приклад розробки найпростішого веб-серверу з використанням фреймворку Flask

Для розробки серверу, який при GET запиті на порт 5000 буде надсилати клієнту HTML сторінку зайняло буквально декілька сторінок. В цьому і є головна філософія фреймворку. Немає необхідності в тому, щоб витрачати на прості задачі багато часу. Наприклад, коли розробнику необхідно додати можливість працювати з БД, фреймворк не нав’язує своє структури чи чогось подібного. Розробник просто може використовувати модулі до яких він звик.

Фреймворк просто надає можливість розширювати власний функціонал за рахунок напрацювань розробника. Немає ніяких обов’язкових моделей, які повинні використовуватись для максимально зручної роботи з додатком. Ось, наприклад, на рисунку 3.2 зображено приклад використання бібліотеки для роботи з MongoDB на сервері:

```
mongo = PyMongo(app)

user_service = UserService(mongo.db.users)
text_service = TextService(mongo.db.texts)

@app.before_first_request
def ensure_admin():
    user_service.ensure_admin_user(ADMIN_EMAIL, ADMIN_PASS)
```

Рисунок 3.2 — Приклад використання інструментів для роботи з MongoDB у Flask

Дуже зручним функціоналом є декоратори фреймворки, які дозволяють виконувати якісь функції на різних життєвих етапах запиту. Це дозволяє, наприклад, виконувати функцію перевірки JWT токену перед кожним запитом, як показано на рисунку 3.3:

```
@app.before_request
def check_auth():
    do_auth = not hasattr(
        app.view_functions[request.endpoint],
        '_allow_without_auth'
    )
    if not do_auth:
        return
    try:
        auth = request.headers.get('Authorization')
        assert auth is not None
        auth_components = auth.split(' ')
        assert len(auth_components) == 2
        token_type, token = auth_components
        assert token_type == 'Bearer'
        result = jwt.decode(token, JWT_SECRET, algorithms=['HS256'])
        g.decoded = result
    except Exception:
        abort(401)
```

Рисунок 3.3 — Приклад використання before_request декоратору для виконання функції перевірки JWT токену перед кожним запитом

Проведений аналіз дозволяє зробити висновок, що веб-фреймворк Flask є прекрасним інструментом для виконання даної задачі.

3.2. Використання середовища розробки Microsoft Visual Studio Code

Середовище розробки Microsoft Visual Studio Code — продукт фірми Майкрософт, який являє собою графічний редактор, модуль роботи з розширеннями, модуль роботи з системою контроля версій Git, засоби рефакторингу, навігації по коду, автодоповнення типових функцій, контекстної підказки і т.ін [22]. Цей продукт дозволяє розробляти як консольні програми, так і програми з графічним інтерфейсом, але найбільше орієнтований на JavaScript розробників. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X. Середовище розробки має дуже зручний інтерфейс. Ось наприклад, на рисунку 3.4 зображено вкладинки для роботи одразу із багатьма файлами, що є дуже зручним на більш-менш великому проект:

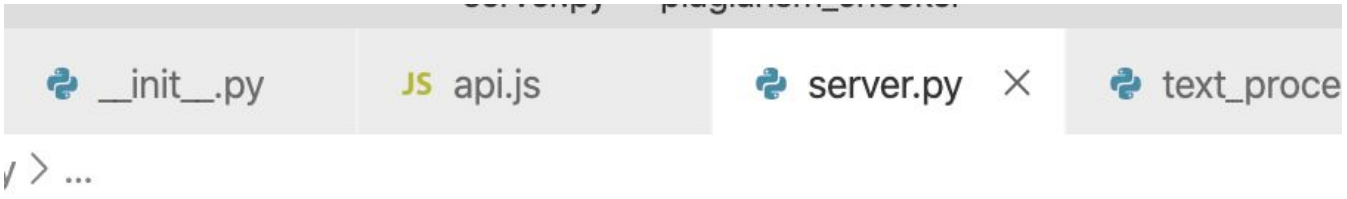

Рисунок 3.4 — Приклад роботи вкладинок у редакторі

Середовище надає дуже багатий функціонал і на одночасному редагуванні багатьох файлів це не закінчується. Редактор надає можливість швидко і зручно працювати із файловою структурою проект, як показано на рисунку 3.5.

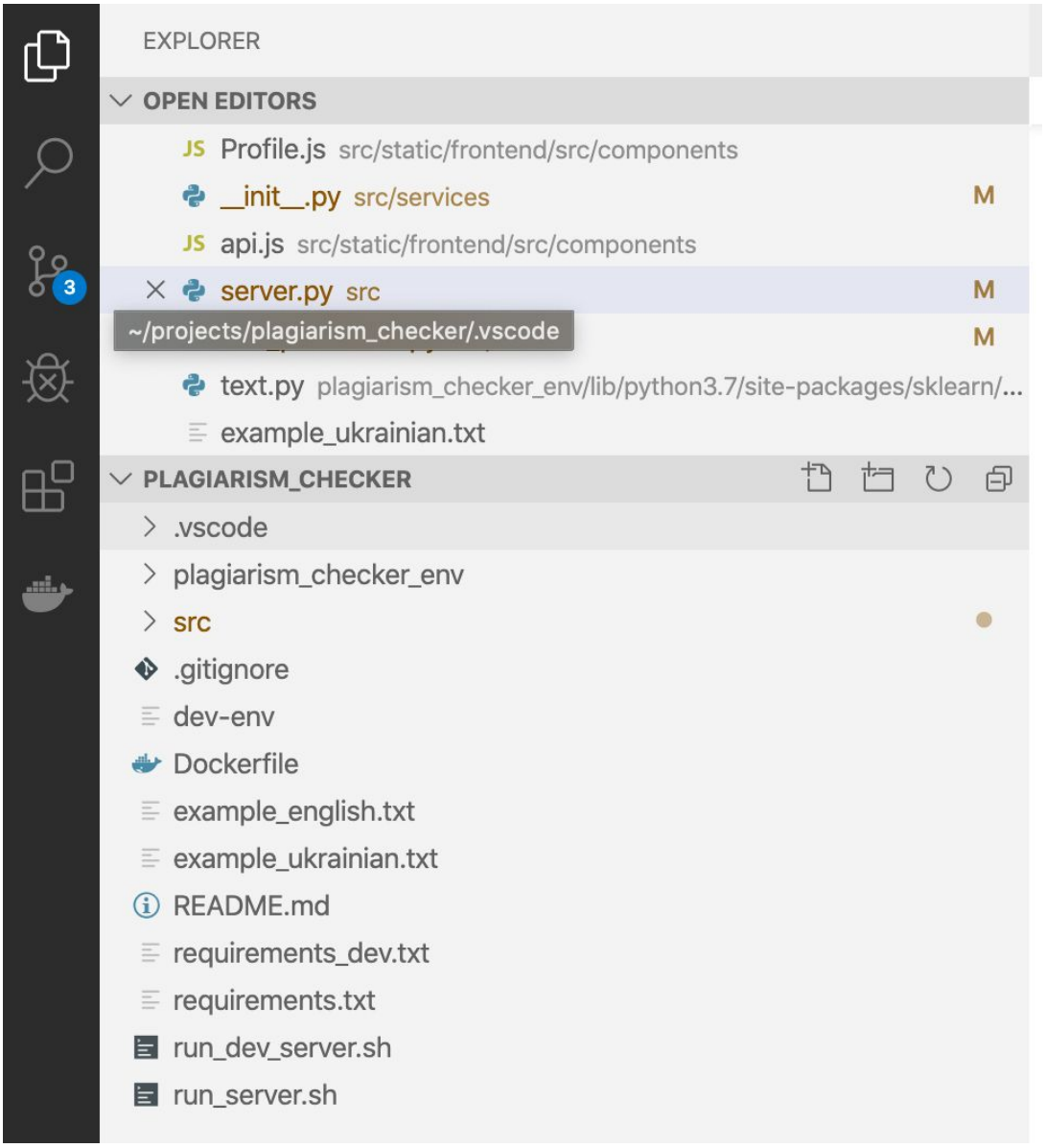

Рисунок 3.4 — Приклад роботи вкладинок у редакторі

Також, що є дуже важливим для розробки даного проекту, Visual Studio code чудово підтримує багато мов. У даному проекті було використано мову Python для розробки серверної логіки і JavaScript для програмної реалізації інтерфейсу користувача. Для обох мов редактор однаково гарно виконував перевірку коректності синтаксису і підсвічування синтаксису. Все це дуже сильно полегшує розробку окремих модулів системи, а значить прискорює саму розробку. Так, наприклад, на рисунку 3.5 зображено приклад обробки редактором одразу Python файлу і JavaScript файлу:

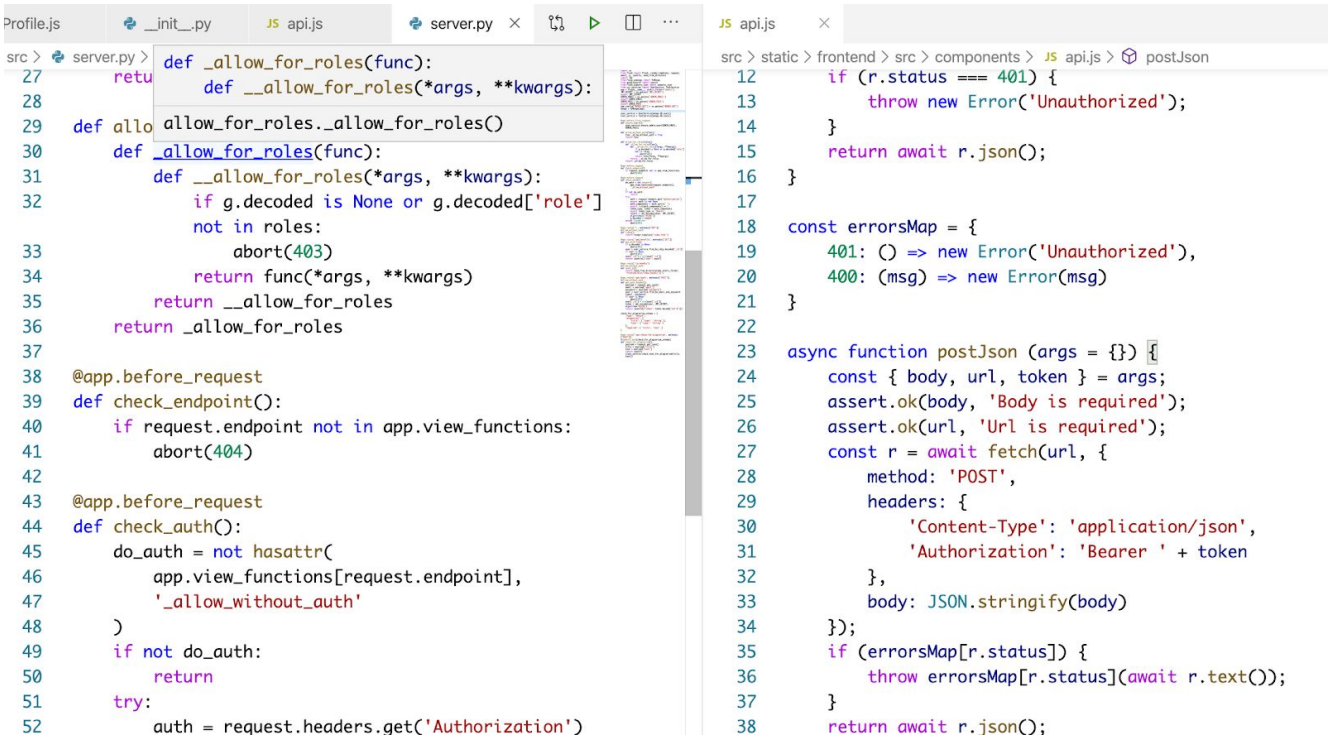

Рисунок 3.5 — Приклад роботи з різними мовами у редакторі

Компанія Microsoft представила Visual Studio Code у квітні 2015 на конференції Build 2015. Це середовище розробки стало першим крос-платформовим продуктом у лінійці Visual Studio. Продукт базується на вільному проекті Atom, що розвивається компанією Github. Зокрема, Visual Studio Code є надбудовою над Atom Shell, що використовують браузерний рушій Chromium і Node.js [20]. Серед підтримуваних мов і технологій: JavaScript, C++, C#, TypeScript, jade, PHP, Python, XML, Batch, F#, DockerFile, Coffee Script, Java, HandleBars, R, Objective-C, PowerShell, Luna, Visual Basic, Markdown, JSON, HTML, CSS, LESS і SASS, Haxe.

3.3. Використання бібліотеки React

Бібліотека React створена для побудови інтерактивних користувацьких інтерфейсів [10]. Продукт розробляється і підтримується компаніями Facebook та Instagram, також має багато вільних розробників. Як завжди у таких мінімалістичних бібліотеках, головний акцент робиться на простоті входу і мінімалістичності коду.

Ось наприклад, на рисунку 3.6 зображено приклад коду, який завантажує всі компоненти для роботи у HTML документі:

src > static > frontend > src > JS index.js

```
1  import React from "react";
2  import ReactDOM from "react-dom";
3  import App from "./components/App.js";
4  import 'regenerator-runtime';
5
6  ReactDOM.render(<App />, document.getElementById("root"));
7
```

Рисунок 3.6 — Приклад головного файлу у React проекті

Бібліотека надає потужні інструменти для розробки інтерфейсів, що включають наступні:

- декларативність — окремі представлення проектуються для кожного стану і React буде займатися їх обробкою і рендером при зміні стану. Декларативність надає коду більшої ясності і полегшує його модифікацію;
- орієнтованість на компоненти — забезпечена можливість створення інкапсульованих компонентів, які самі займаються підтримкою свого стану, та їх композиція для створення складних користувацьких інтерфейсів. Також, враховуючи те, що логіка реалізована мовою JavaScript, а не за допомогою шаблонів, полегшується передача різноманітних даних між компонентами і зникає необхідніть зберігати стану у дереві DOM;
- надає можливість використання створеного один раз компонента у багатьох місцях, що дозволяє додавати новий функціонал без перероблення існуючого коду. Також React може проводити рендер на сервері, використовуючи Node.js і надавати потужності мобільним додаткам, використовуючи React Native.

Існує декілька основних підходи для створення компонентів:

- простий компонент — бібліотека реалізує метод render, який приймає вхідні дані і повертає те, що треба відобразити. Більшість прикладів використовують схожий на XML синтаксис, який називається JSX. Вхідні дані доступні методі render. Синтаксис JSX є необов'язковим для використання у React;
- компонент зі станом — у доповнення до вхідних даних, компонент може підтримувати свої внутрішні дані. Коли стан змінюється, render буде викликаний повторно;
- додаток — використовуючи дані props і state, можна створити самостійний додаток;
- компонент зі сторонніми плагінами — бібліотека є досить гнучкою і надає інструменти, за допомогою яких ви можете працювати зі сторонніми бібліотеками і фреймворками.

Бібліотека React може бути використана для розробки веб та мобільних застосунків. Її мета — забезпечити високу швидкість, простоту та маштабуємість.

3.4. Використання бібліотеки jQuery

Бібліотека jQuery фокусується на взаємодії JavaScript і HTML [12]. Вона дозволяє легко отримувати доступ до елементів DOM, звертатися до атрибутів і вмісту елементів DOM, маніпулювати ними. Також бібліотека jQuery надає зручний API для роботи з AJAX. Продукт підтримується командою jQuery.

Основні можливості бібліотеки:

- має влаштований рушій крос-браузерних CSS-селекторів Sizzle, який виділився у окремий проект;
- робота з деревом DOM;
- події;
- візуальні ефекти;
- доповнення AJAX;

- плагіни JavaScript.

Ця бібліотека дуже довгий час була домінуючою технологією для розробки інтерфейсу користувача, але з часом була витіснена більш новими фреймворками, які краще вирішували проблеми розподілу коду на компоненту і простоту написаного коду. Тим не менш, цей інструмент і досі залишається непоганою альтернативою для маленьких фрагментів коду, коли немає необхідності підключати важкий потужний фреймворк просто для додання якоїсь кнопки. Ось наприклад на рисунку 3.7 приклад додання поведінки при натисненні кнопки:

```
$(document).ready(function(){
  $("p").click(function(){
    $(this).hide();
  });
});
```

Рисунок 3.7 — Приклад додання поведінки для кнопки за допомогою jQuery

3.5. Використання інструмента Webpack

Інструмент розробки Webpack — це збірник JavaScript модулів. Він приймає модулі з їх залежностями як вхідні дані і генерує статичний JavaScript, який готовий до роботи у браузері [23]. Для його роботи необхідний Node.js.

Такі інструменти дозволяють не тільки розширити зручність розробки але і зменшити розмір файлу, який необхідно буде завантажити клієнту для виконання запрограмованого JavaScript проекту. Також Webpack оптимізує деякі нюанси у вихідному коді проекту.

Для того щоб вказати інструменту з якими файлами і як працювати необхідно створити спеціальний файл webpack.config.json, як показано на рисунку 3.8:

```
const path = require("path");

module.exports = {
    entry: "./src/index.js",
    output: {
        path: path.join(__dirname, "/dist"),
        filename: "index-bundle.js"
    },
    module: {
        rules: [
            {
                test: /\.js$/,
                exclude: /node_modules/,
                use: ["babel-loader"]
            },
            {
                test: /\.css$/,
                use: ["style-loader", "css-loader"]
            },
            {
                test: /\.(png|jpg|gif)$/i,
                use: [
                    {
                        loader: 'url-loader',
                        options: {
                            fallback: 'responsive-loader',
                        },
                    },
                ],
            },
        ]
    }
};
```

Рисунок 3.8 — Приклад конфігурації Webpack

3.6. Огляд NoSQL БД MongoDB

База даних MongoDB це розподілена, документо-орієнтована БД загального призначення. Програмний продукт належить до NoSQL баз даних, що означає, що в ньому відсутні поняття звичайних SQL баз даних, такі як: робота з множинами, відношення на множини і т.п. Основна задача MongoDB — це підтримка швидкої пропускної здібності на операції запису та зчитування. Також автор вважає цю БД найзручнішою для швидкої розробки, тому що вона не потребує створення міграцій, все конфігурується у програмному коді.

Складні операції реалізовані за технологією MapReduce. Основний її сенс у тому, щоб як аргумент прийняти всю інформацію що є доступна. Але інформація розподіляється порівну між багатьма екземплярами спеціального типу команд — Map команд. Такі команди приймають фрагмент інформації і повертають новий фрагмент. Результат роботи Map команд обробляють Reduce команди, який приймають багато фрагментів інформації, а повертають одни. Завдяки правильному розподіленню даних між командами і створенню багато екземплярів команд, досягається дуже висока потужність таких операцій. Наприклад, може знадобитися виконати запит на підрахунок кількості користувачів, ім’я яких починається на якусь букву. Для цього Map команда буде отримувати користувача і визначати чи підходить цей користувач під визначений критерій. Якщо так, то вона повертає ідентифікатор даного користувача, якщо ні, то просто поверне null. Далі ці фрагменти інформації будуть оброблені Reduce командою, яка для записів, які будуть містити ідентифікатор буде збільшувати кількість користувачів, а для записів із null значенням — не буде робити нічого.

Одразу при встановленні БД користувачеві буде доступна спеціальний спосіб комунікації із системою — термінал роботи MongoDB. Цей термінал надає можливість виконувати команди пошуку або редагування інформації, отримання

інформації про сам сервер, отримання статистики різних метрик, які веде сама база даних. Ось наприклад, на рисунку 3.9 зображено цей термінал:

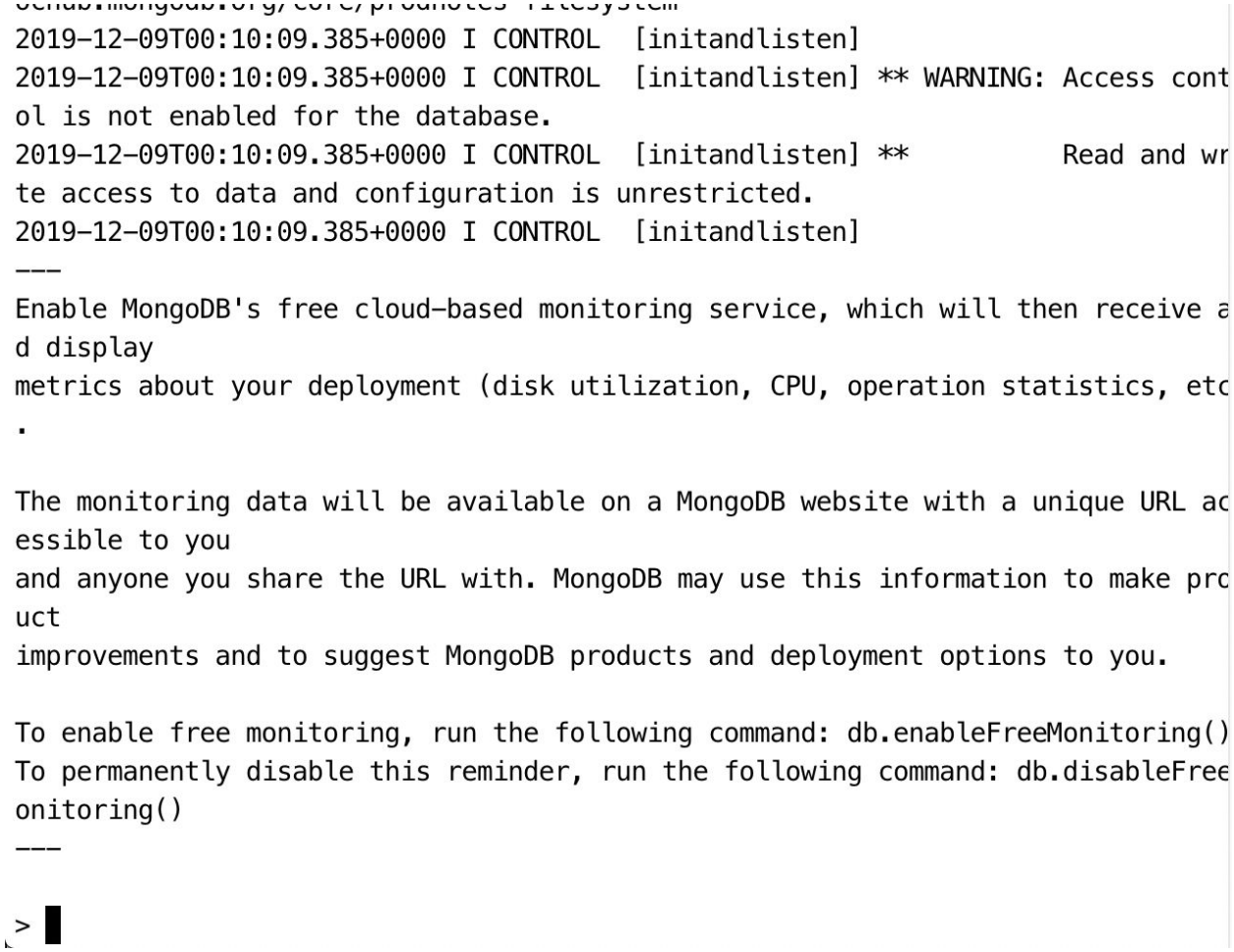

Рисунок 3.9 — Приклад терміналу MongoDB

3.7. Використання інструмента роботи з MongoDB Robo3t

Інструмент Robo3t — це графічний легкий клієнт, що дозволяє відправляти запити до БД MongoDB, та отримувати відповіді у читаємому для людини форматі. Він може працювати як з колекціями, індексами, так і надає користувачеві інтерактивну командну строку де можна вводити команди на виконання.

3.7. Використання інструментів розробки Google Chrome DevTools

Браузер Google Chrome надає чудовий інструмент для розробки та відлагодження користувацьких інтерфейсів Google Chrome DevTools [13]. Їх основна задача сприяти більш швидкому і якісному діагностуванню проблем, що у свою чергу сприяє створенню кращих сайтів. Основні можливості DevTools:

- перегляд та редагування стилів сторінки;
- відлагодження JavaScript;
- перегляд повідомлення та виконання JavaScript у консолі;
- аналіз ефективності використання ресурсів.

Дуже зручним інструментом розробки інтерфейсу користувачі є термінал у Chrome Dev Tools, який зображено на рисунку 3.10:

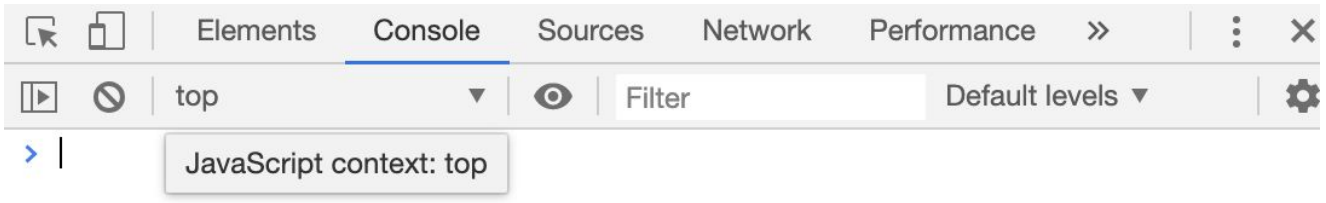

Рисунок 3.10 — Приклад терміналу Chrome Dev Tools

3.8. Огляд програмної платформи Node.js

Інструмент Node.js — це JavaScript-оточення побудоване на JavaScript-рушієві Chrome V8. Розробив це оточення Раян Даль і у 2009 році Node.js був випущений.

Однією з головних переваг платформи у порівнянні з іншими серверними мовами є простота розробки досить складних речей. Ось наприклад, на рисунку 3.11 зображено простий HTTP сервер:

```
const http = require('http')
const port = 3000


const requestHandler = (request, response) => {
    console.log(request.url)
    response.end('Hello Node.js Server!')
}


const server = http.createServer(requestHandler)


server.listen(port, (err) => {
    if (err) {
        return console.log('something bad happened', err)
    }


    console.log(`server is listening on ${port}`)
})
```

Рисунок 3.11 — Приклад програмного коду найпростішого веб-серверу на Node.js

Платформа використовує подієву, неблокуючу Input/Output модель, що робить її легкою та ефективною. Пакетна екосистема Node.js, npm, є найбільшою у світі екосистемою бібліотек з відкритим кодом. Платформа Node.js додає можливість JavaScript взаємодіяти із пристроями вводу-виводу через свій API (написанний на

C++), підключати інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи виклики до них з JavaScript-коду. Інструмент у більшості застосовується на сервері, виконуючи роль веб-серверу, але є можливість розробляти на Node.js і настільні віконні застосунки (за допомогою Electron для Linux, Windows і macOS). Можна навіть програмувати мікроконтроллери.

У платформі все працює за механізмом, який називається Eventloop. Його схематично зображено на рисунку 3.12:

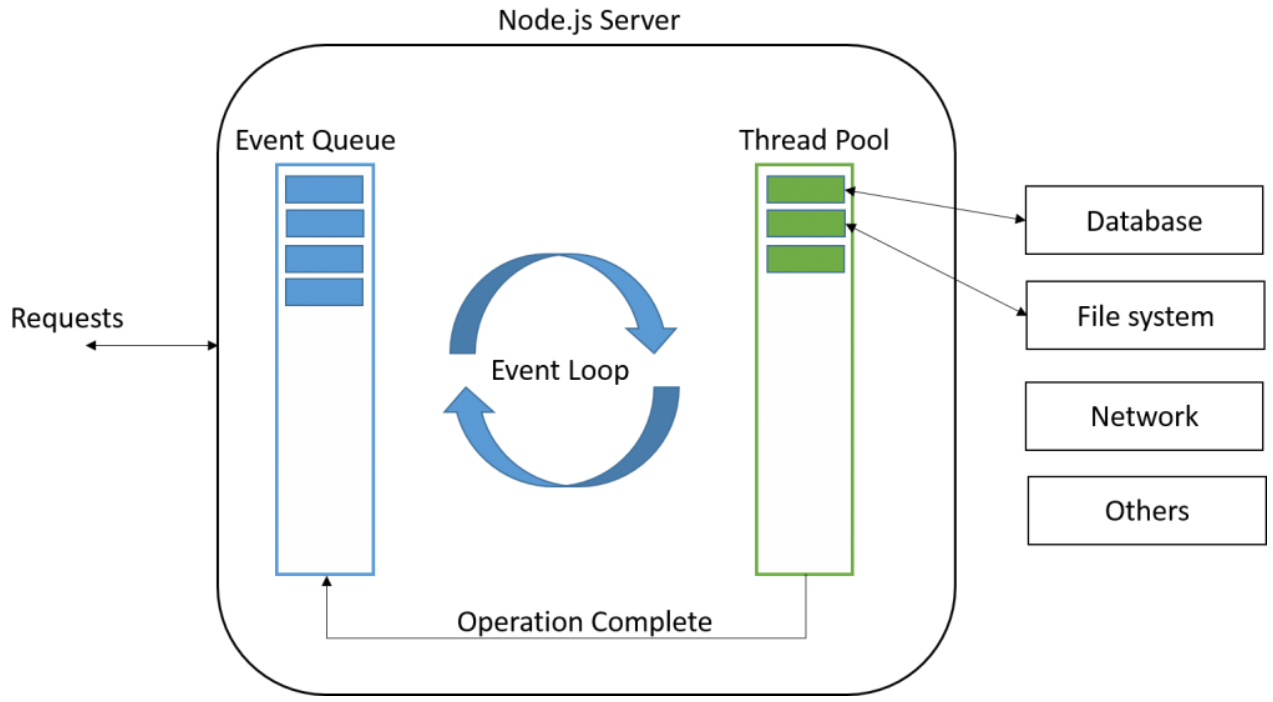

Рисунок 3.12 — Схематичне зображення механізму Event loop у Node.js

Свою популярність платформу завоювала перш за все своєю простотою входу. Для того щоб розробити примітивну програму на Node.js необхідно було лише сам Node.js та створити файл, який містив код. Якщо ж встановлювати які-небудь додаткові модулі, то можна створити спеціальний файл package.json, який буде відслідковувати які бібліотеки потрібно буде встановити при перенесенні програми у інше середовище. Такий підхід дуже сильно розробку продукту великою кількістю

розробників, тому що перенесення коду з одного комп’ютеру на інший реалізовано настільки зручно наскільки це можливо.

Також, якщо розробник вже мав досвід із мовою JavaScript, то він міг витратити на вивчення документації деяку кількість часу і одразу починати створювати досить складні системи. Зі знаннями JavaScript така людина зможе досить повноцінну розробляти серверну частину систему і інтерфейс користувача, що також значно пришвидшує розробку.

В основі Node.js лежить подіє-орієнтоване і асинхронне (або реактивне) програмування з неблокуючим вводом/виводом. Це означає, що жодна операція у файлі, що виконується платформою не може затримати на собі виконання всього скрипту.

Є у платформи, звичайно, і мінуси. Це, наприклад, значно гірша робота із процесорно важкими операціями. Наприклад, якщо програма буде виконувати багато роботи із шифрування розшифруванням, то не тільки не буду ні в чому вигравати у інших серверних мов, а і, можливо, значно поступатися у цьому аспекті. Це все внаслідок однопоточної природи JavaScript. Це означає, що при виконання важкої для процесора операції, усі інші процеси будуть зупинені, доки не закінчить виконання сам процес. Це можна обійти використанням спеціальних “воркерів” які дозволяють виконувати таку роботу в іншому потоці.

Висновок до розділу 3

Розробка системи з клієнт-серверною архітектурою — досить не тривіальна задача. Проте існує достатня кількість різноманітних інструментів, що роблять процес розробки легшим і зрозумілішим.

Фреймворк Flask надає зручне API для реалізації серверної частини з використання HTTP запитів, uyexre структуру файлів і директорій, можливість керування додатком за допомогою тих чи інших розширень.

Бібліотека React надає можливості для створення зручних, інтерактивних користувацьких інтерфейсів. Якщо ж використовувати ще й інструмент Webpack, можна використовувати CommonJs модулі та інші передові можливості мови JavaScript.

Можна зробити висновок, що клієнт-серверна архітектура стала невід’ємною частиною розподілених систем, сфери її застосування постійно збільшуються, що, в свою чергу, стимулює створення інструментів для полегшення розробки таких систем.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

На основі представленої у попередніх розділах інформації видно, що з метою інтерактивної реалізації клієнт-серверної архітектури необхідно реалзіувати певні компоненти. В даному параграфі представлено пояснення схеми роботи створення серверної частини у вигляді Flask додатку і клієнта за допомогою бібліотеки React. Для написання логіки програми використовувалась мова програмування Python [8], який є основою фреймворка Flask, і мова JavaScript для реалізації клієнта.

Узагальнено, розробку системи аналізу збіжності текстової інформації для оцінки плагіату можна представити сукупністю задач, які можна поділити на дві основних частини: задача, що полягає у створенні серверу, що міститиме основну логіку та задача, що полягає у створенні клієнту, що забезпечуватиме комфортну роботу із застосунком.

4.1. Реалізація програмного продукту

Перш за все слід розглянути опис програмного продукту у діаграмах. На рисунку 4.1 зображено діаграму прецедентів додатку:

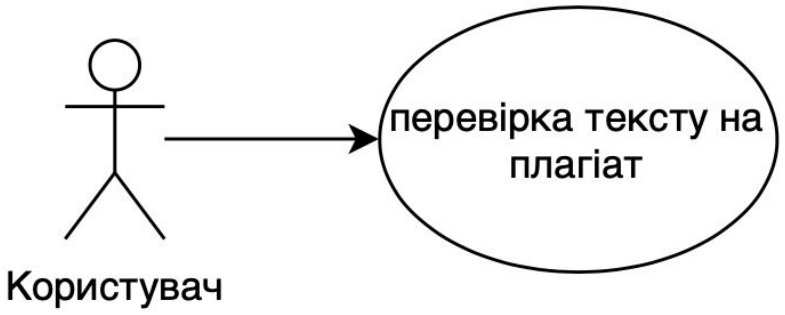

Рисунок 4.1 — Діаграма прецедентів

Функціонал системи досить вузький, орієнтований більше на якість виконуваної роботи. Користувачу також необхідно небагато кроків для використання додадку як зображено на діаграмі послідовності на рисунку 4.2:

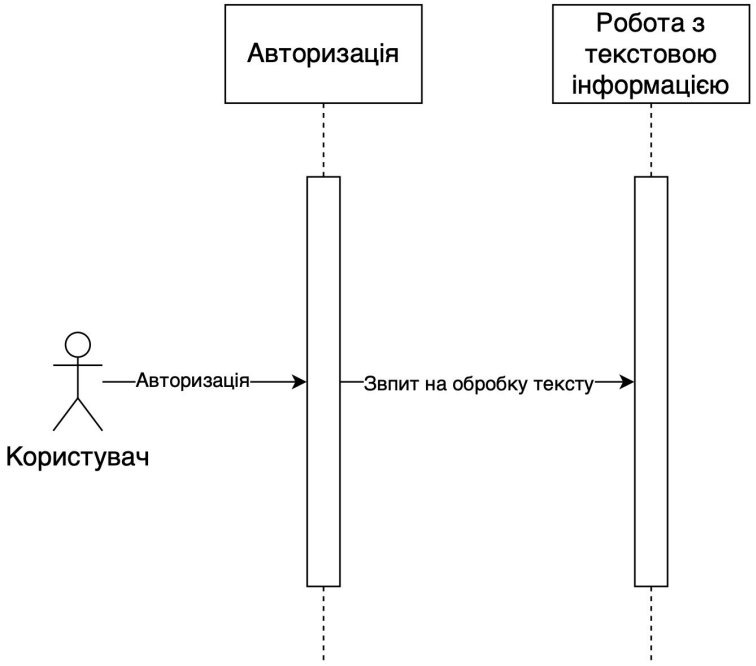

Рисунок 4.2 — Діаграма послідовності

Під час авторизації, користувач надсилає JWT (JSON Web token) ключ для ідентифікації. Після цього він матиме змогу працювати з формою перевірки тексту. Для запуску процесу перевірки тексту клієнт надсилає POST запит з JSON [3] інформацією, що включає заголовок та саму текстову інформацію. Слід відзначити REST-full структуру серверної частини: для забезпечення авторизованого доступу до функціоналу у заголовках запиту клієнт має надсилати JWT токен, за яким сервер може ідентифікувати користувача. У звичайних реалізаціях, зазвичай, використовуються сесії, які підтримують стан користувачів: а саме хто коли зареєструвався, авторизувався і т.п.

Підхід же з JWT токенами позбавляє необхідності підтримувати складну логіку сесій авторизації. Безпека забезпечується тим, що цей токен надсилається у

зашифрованому вигляді і лише сервер має ключ, яким токен може бути розшифрованим. Під час авторизації, користувач надсилає свої логін та пароль, а у відповідь отримує зашифрований токен, який містить інформацію про користувача та дату дії.

Компоненти серверу виконані у вигляді Python файлів та має таку структуру, як зображено на рисунку 4.3:

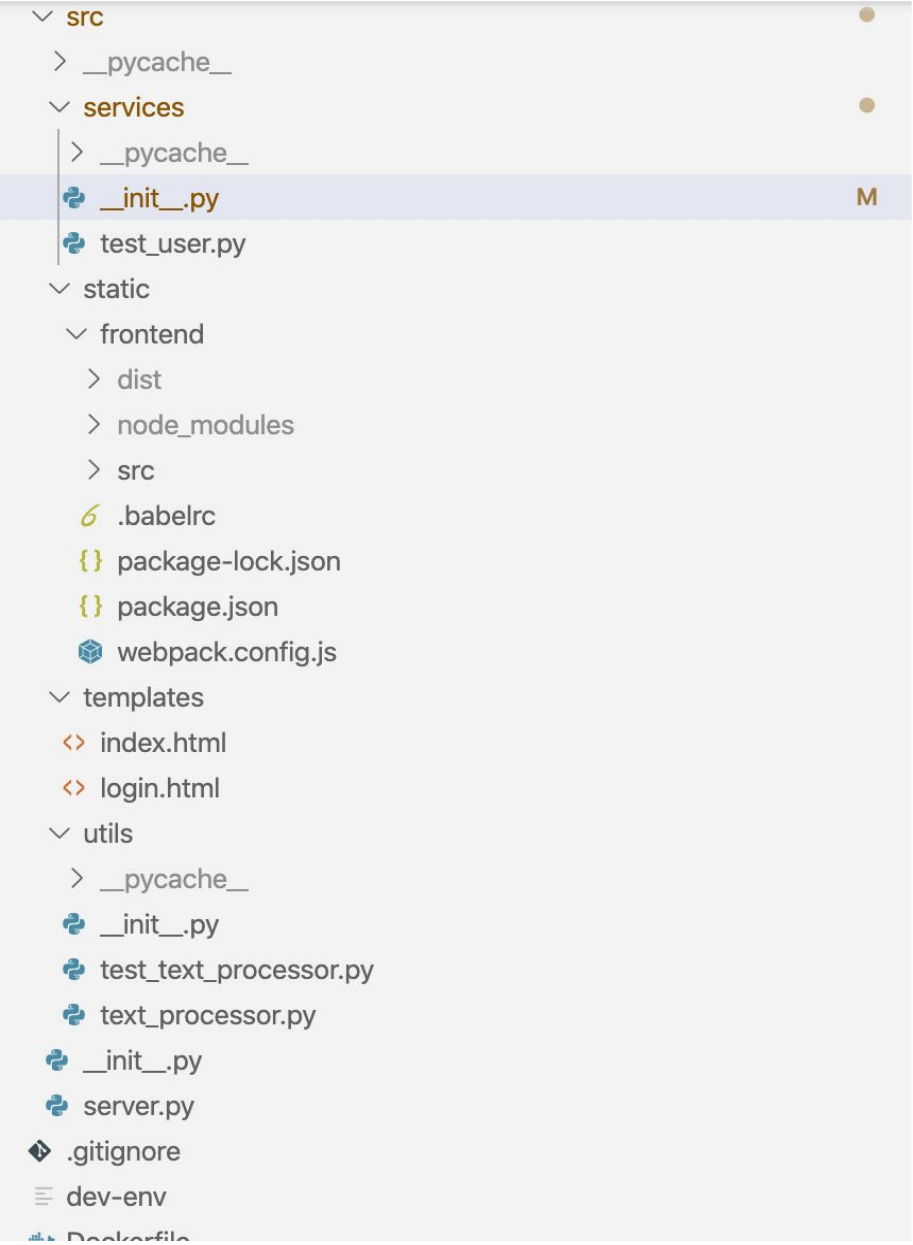

Рисунок 4.3 — Структура проекту

Фреймворк Flask не примушує ніякої своєї структуру тому архітектура проекту залежить від побажань автора. Ось наприклад який вигляд має сервіс роботи з користувачами у БД на рисунку 4.4:

```
class Service:
    def __init__(self, collection):
        self.coll = collection

class UserService(Service):
    def find_by_email_and_password(self, email, password):
        user = self.coll.find_one({'email': email})
        if user is None:
            return None
        expected_pass = password.encode('utf-8')
        actual_pass = user['password'].encode('utf-8')
        if bcrypt.checkpw(expected_pass, actual_pass):
            return user
        return None

    def ensure_admin_user(self, email, password):
        hashed = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
        admin = {
            'first_name': 'Admin',
            'last_name': 'Admin',
            'email': email,
            'password': hashed.decode('utf-8'),
            'role': 'admin'
        }
        self.coll.update_one({'email': email}, {'$set': admin}, upsert=True)

    def find_by_id(self, id):
        return self.coll.find_one({'_id': ObjectId(id)})
```

Рисунок 4.4 — Приклад сервісу для роботи з БД

У сервісах роботи з БД реалізований базовий клас, який зберігає у себе в стані об’єкт для роботи з відповідною колекцією MongoDB. Так наприклад, сервіс для роботи з користувачами ініціюється з об’єктом колекції users. Це дозволяє

наслідувати класи від базового і не повторювати кожного разу логіку збереження об’єкту колекції.

Для роботи з текстами було реалізовано додатковий модуль text_processor.py як показано на рисунку 4.5:

```
import operator
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def get_topn_words(text, n):
    vectorizer = CountVectorizer()
    vectorizer.fit([text])
    feature_names = vectorizer.get_feature_names()
    vector = vectorizer.transform([text]).toarray()[0]
    result = []
    word_map = {v: vector[k] for k, v in enumerate(feature_names)}
    for _ in range(n):
        the_most_common_word = max(word_map.items(), key=operator.itemgetter(1))[0]
        result.append(the_most_common_word)
        del word_map[the_most_common_word]
    return result

def get_cosine_sim(strs):
    vectors = [t for t in get_vectors(strs)]
    return cosine_similarity(vectors)

def get_vectors(strs):
    vectorizer = CountVectorizer(ngram_range=(1, 3))
    return vectorizer.fit_transform(strs).toarray()
```

Рисунок 4.5 — Реалізацію модулю для роботи з текстом

У даному прикладі, використовуються декілька функцій: get_vectors для створення векторів з набору текстів; get_cosine_sim для отримання матриці коефіцієнтів збіжності переданих текстів. Така структура допомагає легше використовувати даний функціонал у проекті.

Не менш важливим моментом, є можливість написання автоматичних unit тестів до програмних модулів як показано на рисунку 4.6:

```
from unittest import TestCase, main
from .text_processor import get_topn_words, get_cosine_sim

class TestGetTopNWords(TestCase):
    def test_should_return_top3_w(self):
        text = 'AI and humans have always been friendly with AI AI AI and a
        actual = get_topn_words(text, 3)
        self.assertListEqual(actual, ['ai', 'and', 'have'])

class TestGetCosineSim(TestCase):
    def test_should_return_cosine_sim(self):
        texts = [
            'AI is our friend and it has been friendly',
            'AI and humans have always been friendly with AI AI AI and and
            'AI is not a human'
        ]
        actual = get_cosine_sim(texts)[0]
        self.assertAlmostEqual(actual[0], 1, 2,  '100 percent equality')
        self.assertAlmostEqual(actual[1], 0.5, 1, '50 percent equality')
        self.assertAlmostEqual(actual[2], 0.3, 1, '30 percent equality')

    def test_check_ukrainian_text(self):
        texts = [
            'Великого значення в цей час набули церемоніали, етикет, ушляхе
            людини.',
            'У цей час великого значення набув етикет, покращення способу ж
        ]
        actual = get_cosine_sim(texts)[0]
        self.assertAlmostEqual(actual[0], 1, 2,  '100 percent equality')
        self.assertAlmostEqual(actual[1], 0.5, 1, '50 percent equality')

if __name__ == '__main__':
    main()
```

Рисунок 4.6 — Приклад автоматичного тесту для модулю text_processor

Автоматичні тести дозволяються прискорити швидкість розробки. Прискорення досягається тим, що кожен раз дописуючи новий функціонал, не потрібно перевіряти на коректність вже покриті тестами модулі, а можна просто запустити автоматичні тести і переглянути результат. Також це дуже допомагає при автоматичній побудові проекту під час додання нового функціоналу у систему контролю версій. Наприклад, у Git при доданні нових комітів, можна додати функціонал запуску тестів, який попередить завантаження непрацюючого коду у центральний репозиторій [27].

Для відображення початкової веб-сторінки системи, або форми авторизації, необхідно якимось чином отримати їх. У браузері всі сторінки є HTML-сторінками. Мова розмітки HTML (HyperText MarkUp Language) — це єдина мова формування HTML веб-документів.

При формуванні модулю серверу, можна вказати різноманітні опції для реалізації необхідного функціоналу. Наприклад, можна задати щоб якісь операції виконувались один раз перед першим запитом до серверу, як показано на рисунку 4.7:

```
text_service = TextService(mongo.db.texts)

@app.before_first_request
def ensure_admin():
    user_service.ensure_admin_user(ADMIN_EMAIL, ADMIN_PASS)
```

Рисунок 4.7 — Приклад виконання операції перед першим запитом до серверу

Схожим чином можна програмувати не тільки операції перед першим запуском, а і більш складніші речі. Наприклад, перевірку JWT токену необхідно робити перед кожним запитом. Це може бути реалізовано як показано на рисунку 4.8:

```python
@app.before_request
def check_auth():
    do_auth = not hasattr(
        app.view_functions[request.endpoint],
        '_allow_without_auth'
    )
    if not do_auth:
        return
    try:
        auth = request.headers.get('Authorization')
        assert auth is not None
        auth_components = auth.split(' ')
        assert len(auth_components) == 2
        token_type, token = auth_components
        assert token_type == 'Bearer'
        result = jwt.decode(token, JWT_SECRET, algorithms=['HS256'])
        g.decoded = result
    except Exception:
        abort(401)
```

Рисунок 4.8 — Приклад реалізації перевірки JWT токену перед кожним запитом

Слід відзначити, що відбувається, коли функція отримує помилку при спробі розшифрувати JWT токен. Таке може статися, щонайменше, у 2 випадках: коли JWT токен зашифрований іншим ключем; у JWT токена вийшов час, коли він був коректним.

Розшифрований JWT токен є не чим іншим як об’єктом словником, що містить інформацію про користувача. Цю інформацію можна використовувати різними способами: реалізовувати додаткову логіку в залежності від користувача; реалізацію контролю доступу до ресурсів за правами доступу конкретного користувача; просто мати доступ до інформації про користувача в усіх частинах модулю серверу.

Для того, щоб перевірити чи правильний токен збережений у браузері, клієнт надсилає його на сервер і очікує отримати інформацію про даного користувача.

Сервер, отримавши запит на відповідний шлях, перевіряє розшифрований токен і робить запит до БД, щоб дістати актуальну інформацію як показано на рисунку 4.9:

```
@app.route('/api/profile', methods=['GET'])
def get_profile():
    if g.decoded is None:
        abort(401)
    user = user_service.find_by_id(g.decoded['_id'])
    if user is None:
        abort(401)
    user['_id'] = str(user['_id'])
    return jsonify({'user': user})
```

Рисунок 4.9 — Функція яка повертає інформацію про користувача

Важливою частиною серверного додатку є коректна поведінка у непередбачуваних ситуаціях. Наприклад, користувач відправив запит із шляхом, який не є реалізованим на сервері. У такому випадку, потрібно зупинити виконання усієї додаткової користувацької логіки і видати код помилки 404, як показано на рисунку 4.10:

```
@app.before_request
def check_endpoint():
    if request.endpoint not in app.view_functions:
        abort(404)
```

Рисунок 4.10 — Функцій яка перевіряє коректність шляху у запиті

Необхідно також описати функцію яка приймає запит на перевірку тексту. Ця функція включає функціонал валідації тіла запиту. В даному випадку використовується POST запит. Це дозволяє передавати інформацію у JSON форматі безпечніше ніж така сама передача даних через параметри GET запиту. Для того

щоб сервер викликав необхідну функцію, її треба помітити спеціальними Python декораторами, як показано на рисунку 4.11:

```
check_for_plagiarism_schema = {
    'type': 'object',
    'properties': {
        'title': { 'type': 'string' },
        'text': { 'type': 'string' }
    },
    'required': [ 'title', 'text' ]
}

@app.route('/api/check-for-plagiarism', methods=['POST'])
@expects_json(check_for_plagiarism_schema)
def check_for_plagiarism():
    payload = request.get_json()
    title = payload['title']
    text = payload['text']
    return jsonify(text_service.check_text_for_plagiarism(title, text))
```

Рисунок 4.11 — Функція яка запускає перевірку тексту

Однією з важливих складових успішної розробки програмного забезпечення є керування складністю. У даному випадку, керування складністю полягає у відокремленні логіки серверу по обробці HTTP запиту від логіки роботи з БД. У даному випадку було вирішено використати підхід окремих класів-сервісів. Такий підхід проектування програмного забезпечення полягає в тому, щоб створювати окремі класи, які будуть надавать методи для роботи з БД, такі як: отримання інформації з БД; запис у БД. Самі ж ці класи-сервіси буду інкапсулювати програмний код для роботи з БД, у даному випадку MongoDB. Основною перевагою такого підходу є покращена зрозумілість коду. Коли програміст читатиме реалізацію функції він бачитиме, наприклад, логіку роботи з HTTP запитом і лише декларативний виклик у класу-сервісу. Якщо підібрати гарну назву для методу,

можливо, не буде необхідності вивчати реалізацію саме цього методу, адже це буде зрозуміло з назви. Всі ці невеликі деталі виглядають незначними, але значно покращують швидкість розробки і якість підтримки програмного забезпечення. Приклад класу-сервісу приведено на рисунку 4.12:

```
class UserService(Service):
    def find_by_email_and_password(self, email, password):
        user = self.coll.find_one({'email': email})
        if user is None:
            return None
        expected_pass = password.encode('utf-8')
        actual_pass = user['password'].encode('utf-8')
        if bcrypt.checkpw(expected_pass, actual_pass):
            return user
        return None

    def ensure_admin_user(self, email, password):
        hashed = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
        admin = {
            'first_name': 'Admin',
            'last_name': 'Admin',
            'email': email,
            'password': hashed.decode('utf-8'),
            'role': 'admin'
        }
        self.coll.update_one({'email': email}, {'$set': admin}, upsert=True)

    def find_by_id(self, id):
        return self.coll.find_one({'_id': ObjectId(id)})
```

Рисунок 4.12 — Клас-сервіс для роботи з колекцією користувачів

Зазвичай, ієрархія проекту включає більшу кількість різних компонентів. Наприклад, у цибулевій архітектурі є контролери, сервіси та репозиторії. Але у даному випадку основна складність проекту полягала у алгоритмі, тому було вирішено не надто розвивати тему архітектури. Наслідком такого рішення, є те, що у сервісах може міститися не тільки логіка роботи з БД, а й інша логіка. Наприклад у

класі TextService використано функції для роботи з аналізом збіжності текстів, як показано на рисунку 4.13:

```
def check_text_for_plagiarism(self, title, text):
    links = self.get_topn_links(title, LINKS_TO_CHECK)
    texts_to_check = [item for item in map(
        lambda l: l.text, links
    )]
    # in order to get similarity to first text insert it here
    texts_to_check.insert(0, text)
    plagirism_coefs_array = get_cosine_sim(texts_to_check)[0]
    texts_to_check = texts_to_check[1:]
    plagirism_coefs_array = plagirism_coefs_array[1:]

    return [{
        'url': links[i].url,
        'text': v,
        'title': links[i].page_title,
        'plagiarism_coef': plagirism_coefs_array[i]
    } for i, v in enumerate(texts_to_check) if plagirism_coefs_array[i] >= 0.2]
```

Рисунок 4.13 — Метод для аналізу збіжності текстів

Також слід відзначити і метод для отримання посилань на відкриті джерела для завантаження текстів. Метод описано на рисунку 4.14.

Враховуючи все вищесказане, можна отримати приблизну картину роботи серверної частини додатку. Проте жодна серверна частина не може бути корисною, якщо результат її роботи не може використати користувач, тим чи іншим способом. Тому необхідно було забезпечити зв’язок між користувачем і серверною частиною. Тут було використано клієнт-серверну архітектуру і один із її стилів — REST. Це означає, що весь функціонал серверної частини залишається, але викликається у відповідь на HTTP запити від клієнта. Веб-клієнт запускається у браузері користувача і після логін форми (рисунок 4.15), користувач попадає на головну сторінку.

```
def get_topn_links(self, seach_str, Nlinks):
    result = []
    for url in search(seach_str, stop=Nlinks):
        try:
            response = urllib.request.urlopen(url).read()
            html = response.decode('utf-8')
            soup = BeautifulSoup(html, 'html.parser')
            text = soup.find_all(text=True)
            output = ''

            for t in text:
                if t.parent.name not in bs4_tags_blacklist:
                    output += '{} '.format(t)
            link = Link(
                url=url,
                text=output,
                raw_html=html,
                page_title=soup.title.string
            )
            result.append(link)
        except Exception as e:
            print('Got exception')
            print(e)
    return result
```

Рисунок 4.14 — Метод для отримання посилань на відкриті джерела

Модуль пошуку інформації у відкритих джерелах у даній реалізації використовує пошукову систему Google, але може бути легко адаптований до іншої пошукової системи, або навіть електронної бібліотеки. Після отримання посилання на джерело модуль завантажує HTML документ і за допомогою бібліотеки beautifulsoup виокремлює зміст джерела [28]. Завдяки цьому, можна отримати текст статті для використання його в подальшій обробці. Для подальшого використання, формується список посилань разом із відповідним текстом посилання, який і є значенням, яке повертає модуль. Використовуючи вищезгадану структуру, можна отримати досить точну інформації про збіжність текстів із фрагментом, що надається на перевірку.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Login</title>
</head>
<body>
    <form id="login-form">
        <input id="email" type="email" name="email"/><br/>
        <input id="password" type="password" name="password"/><br/>
        <button id="btn-submit" type="submit">Submit</button>
    </form>
    <script>
        async function postData(email, password) {
            const result = await fetch('/api/auth', {
                method: 'POST',
                headers: {
                    'content-type': 'application/json'
                },
                body: JSON.stringify({
                    email, password
                })
            });
            const body = await result.json();
            console.log('Done with body', body);
            localStorage.setItem('jwt-token', body.token);
            document.location.href = '/';
        }
        window.onload = () => {
            const btn = document.getElementById('btn-submit');
            btn.onclick = (e) => {
                e.preventDefault();
                const email = document.getElementById('email').value;
                const password = document.getElementById('password').value;
                postData(email, password);
            }
        };
    </script>
</body>
</html>
```

Рисунок 4.15 — HTML сторінка авторизації

Коли користувач опиняється на головній сторінці, запускається скрипт, який ініціює React компоненти і далі інтерфейс користувача коригується цією бібліотекою. Це відбувається за допомогою завантаження Javascript скрипту у головній HTML сторінці [10]. Перш за все, було створено головний компонент App.js і його render функція зображена на рисунку 4.16:

```
render() {
    return (
        <div>
            { this.state.isLoading
                ? <Loader />
                : this.state.isLoggedIn
                    ? <ProfilePage checkForPlagiarism={this.checkForPlagiarism.bind(this)}/>
                    : <LoginPage updateToken={this.updateToken.bind(this)} /> }
        </div>
    );
}
```

Рисунок 4.16 — Метод render компоненту App

Бібліотека React надала можливості перевикористання програмного коду та полегшила розбиття всього на модулі. Так можна побачити, що головний компонент App відображає різні компоненти. Наприклад, коли дані загружаються, відображається компонент Loader. Цей компонент представляє собою gif анімацію обертання кола. Завдяки йому, користувач може бачити що додаток щось опрацьовує в даний момент. Далі вже в залежності від того, авторизувався користувач або ні, йому відображається компонент ProfilePage або ж LoginPage. Такий спосіб розділення логіки авторизації користувача від функціонального компоненту сприяє спрощенню написання програмного коду. Функціональному компоненту не має необхідності перевіряти чи користувач авторизований, чи він має права використовувати той чи інший функціонал. Таку задачу повинні виконувати компоненти, які відображають ті чи інші функціональні компоненти. Використання бібліотеки React також полегшує нам створення динамічного контенту на сторінці. Наприклад, на рисунку 4.17 зображено відображення посилань на відкриті джерела, що були отримані з серверу:

```
render() {
    const { output, isLoading, error } = this.state;
    return (
        <div>
            Головна сторінка <br/>
            <OptionalAlert msg={error}/>
            <input
                type="text"
                className="form-control"
                name="title"
                value={this.state.title} onChange={this.onChange.bind(this)}
            /><br/>
            <div className="input-group">
                <Textarea
                    className="form-control"
                    aria-label="With textarea"
                    name="text"
                    onChange={this.onChange.bind(this)}
                >{this.state.text}</Textarea>
            </div>
            <button
                className="btn btn-primary"
                onClick={this.onSubmit.bind(this)}
            >Підтвердити</button>
            <br/>
            {
                isLoading
                    ? <Loader />
                    : <div>{output.map((e, i) =>
                        <PlagiarismResultItem
                            key={i}
                            number={i}
                            link={e}
                        ></PlagiarismResultItem>)}
                      </div>
            }
        </div>
    );
}
```

Рисунок 4.17 — Відображення посилань, отриманих з серверу

Висновок до розділу 4

У даному розділі було наведено опис реалізації програмного продукту. Була представлена діаграма прецедентів та описані основні сценарії використання продукту, а також їх програмна реалізація. Було описано створення серверної частини засобами Flask, а саме було використано такі компоненти як request, render_template, abort, g, jsonify, використано модуль PyMongo, та засоби авторизації. Також було описано створення користувацького інтерфейсу, який відображається під час роботи системи.

Для реалізування клієнтської частини було використано бібліотеку React. З її допомогою було реалізовано гнучкий і зручний інтерфейс, можливість робити HTTP запити до серверу і надано інтерфейсу інтерактивності. Використання інструменту Webpack дало можливість використання CommonJs модулів, що в свою чергу полегшує розробку, і дає можливість використання npm модулів. Для виконання HTTP запитів до серверу був використаний модуль axios, що інкапсулює логіку використання влаштованого у всі браузери XmlHttpRequest.

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для забезпечення безвідмовної роботи програмної системи контролю доручень треба дотримуватися основних вимог при інсталяції та рекомендацій щодо її використання.

5.1. Системні вимоги та запуск

Для запуску розробленої програмної системи персональний комп'ютер повинен мати двох-ядерний процесор Intel ® чи AMD з тактовою частотою не нижче 2.0 GHz, відеокарта серії NVIDIA GeForce 470 GT або ж вбудоване у процесор графічне ядро, 4 ГБ оперативної пам'яті, на комп'ютері повинна бути встановлена операційна система Ubuntu, MacOS, Windows 7, Windows 8, Windows 10, а також фреймворк Flask 1.0.3 та бібліотеки: numpy, sklearn, pyjwt, bcrypt, pymongo, Flask-PyMongo, beautifulsoup4 і flask_expects_json. Необхідно також встановлена БД MongoDB. Також на жорсткому диску повинно бути не менше 300 Мб вільного місця.

Для запуску системи необхідно запустити виконавчий файл run_dev_server.sh (рисунок 5.1).

```
(plagiarism_checker_env) ➜  plagiarism_checker git:(feature-21_setup_instructions) ✗ ./run_dev_server.sh
 * Serving Flask app "./src/server.py" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 253-509-234
```

Рисунок 5.1 — Запуск виконавчого файлу

Після цього можна відкрити браузер і за адресою http://127.0.1:5000 можна відкрити інтерфейс системи (рисунок 5.2).

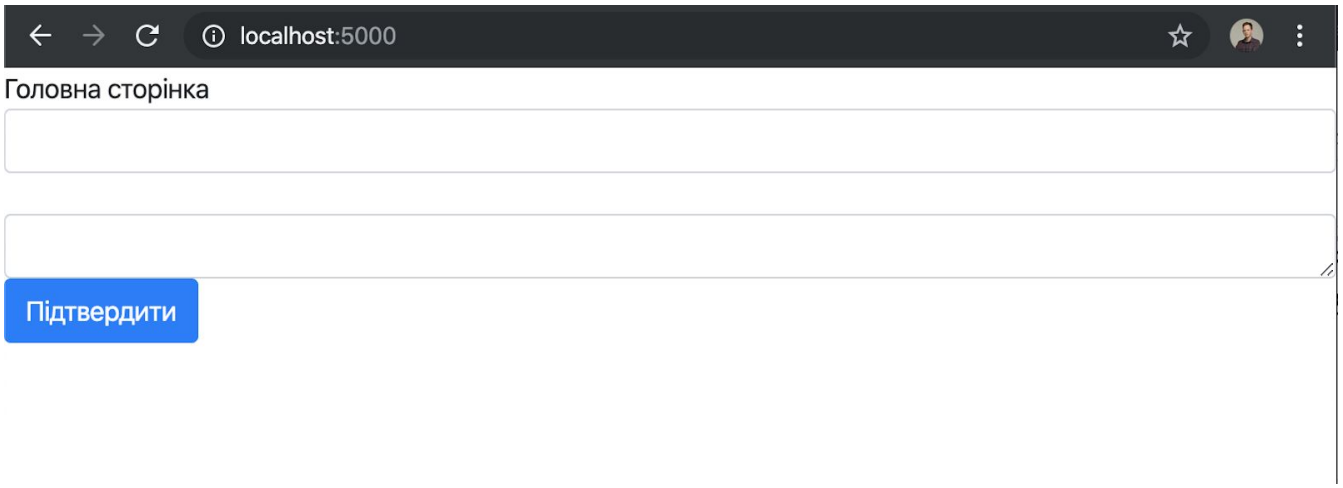

Рисунок 5.2 — Головна сторінка

Якщо користувач зареєстрований але зайшов з іншого браузеру або ж срок його сесії закінчився, він може авторизуватися на сторінці авторизації (рисунок 5.3).

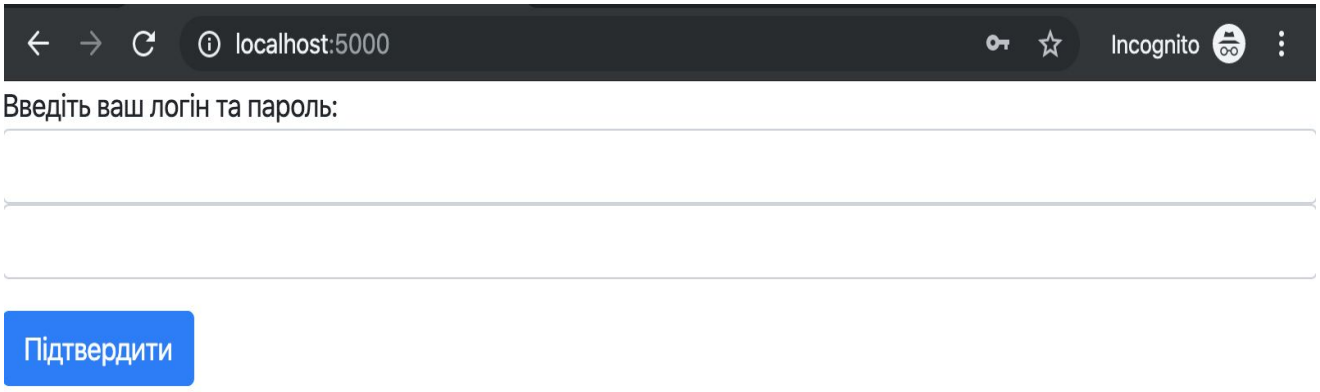

Рисунок 5.3 — Сторінка авторизації

5.2. Перевірка текстової інформації

Керування додатком виконується за допомогою клавіатури та мишки. Для перевірки текстової інформації користувачеві необхідно ввести у поле заголовку тему або тематику тексту. Це необхідно для визначення за якою тематикою шукати відкриті джерела. Далі необхідно ввести текст, який необхідно перевірити у поле для тексту. Приклад заповнення показано на рисунку 5.4:

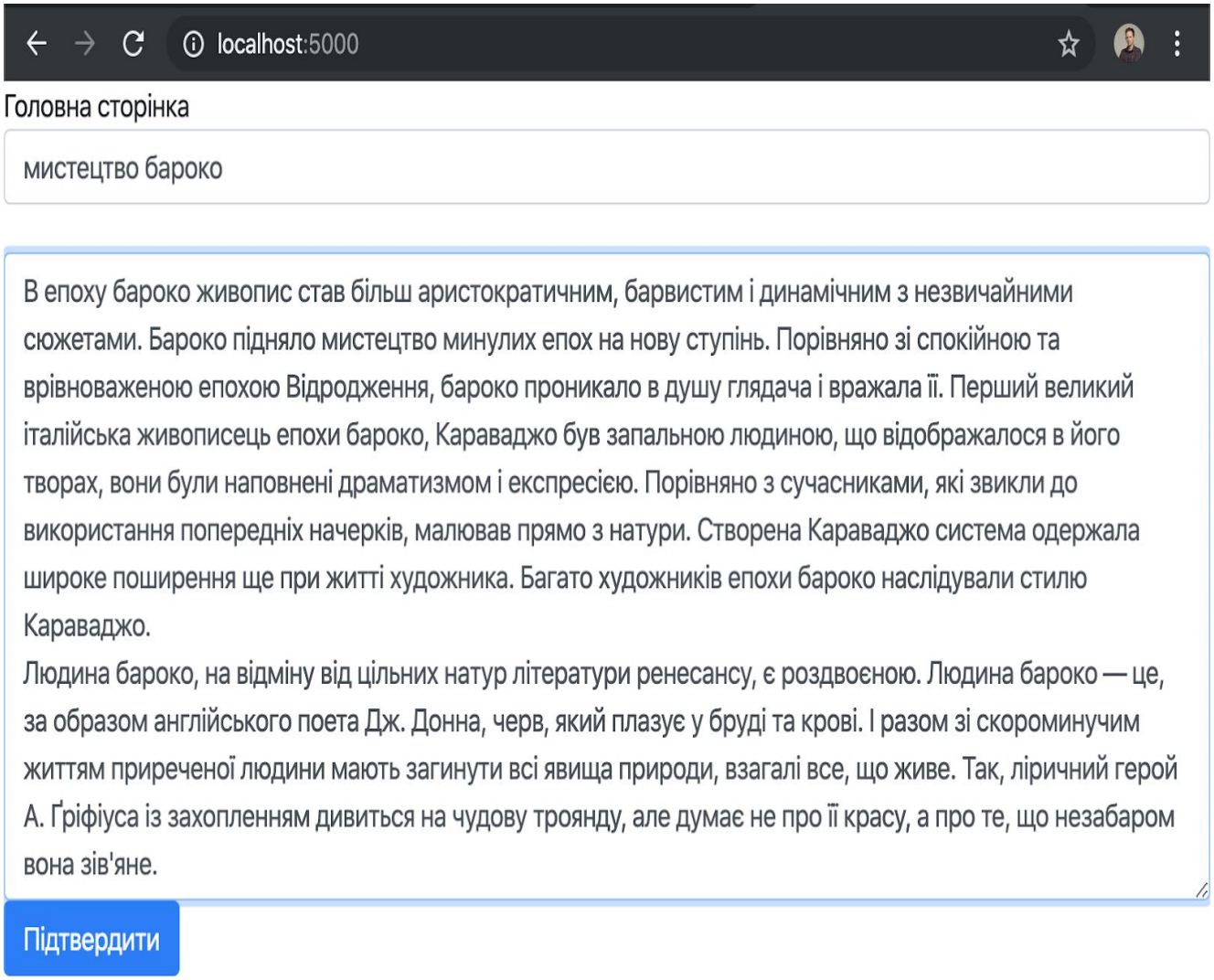

Рисунок 5.4 — Приклад заповнення форми для перевірки тексту

Після заповнення форми необхідно натиснути кнопку “Підтвердити” для того, щоб запустити пошук. У результаті буде відображено анімацію завантаження інформації, як показано на рисунку 5.5:

Рисунок 5.5 — Покажчик завантаження інформації

Такий принцип побудови додатку забезпечує те, що користувач не буде відчувати моментів, коли система можливо довго виконуватиме запит, або ще щось таке. При натисканні кнопки для запуску якоїсь обробки, одразу користувачеві буде зображено анімацію виконання роботи. Таким чином, користувач сприймає систему швидшою і плавнішою ніж вона насправді може бути. Такі дрібниці дуже сильно впливають на глобальне сприйняття продукту. Якщо інтерфейс буде повільно відповідати на дії користувача, то і сама система користувачеві буде здаватися повільною.

Таке ускладнення інтерфейсу спричинено переходом усіх веб-додатків на клієнт-серверну архітектуру, де необхідно теперь якось показувати клієнту, що відбувається якась робота. Адже користувач може подумати, що із системою щось не так і що вона на відповідає його задачам по швидкості.

Після завантаження, інформацію буде відображено у вигляді списку посилань під формою заповнення тексту (рисунок 5.6).

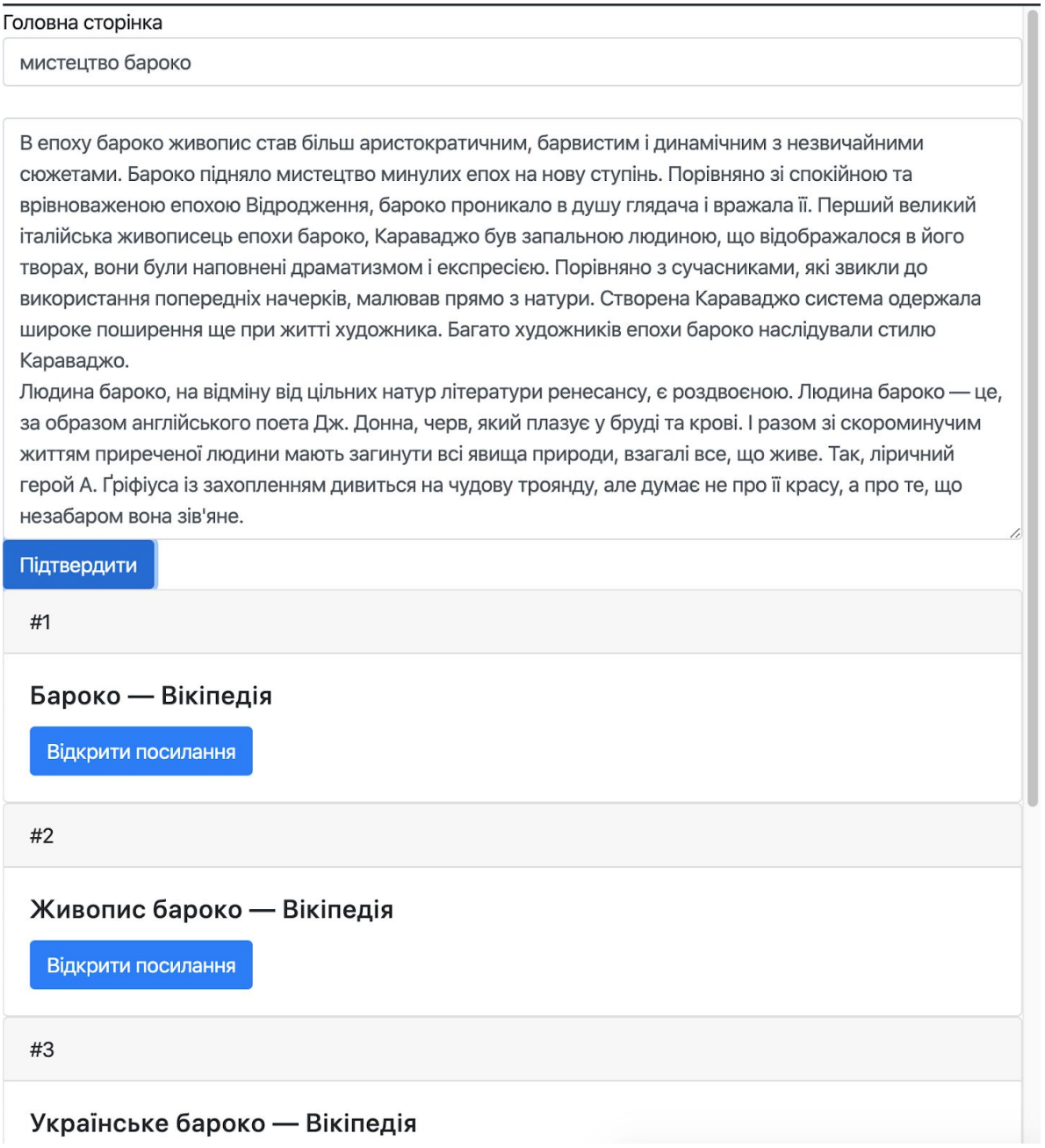

Рисунок 5.6 — Результат перевірки тексту

Результат перевірки включає посилання на відкриті джерела, які були визначені як досить схожі текст, що перевіряється. Для того, щоб відкрити посилання, необхідно натиснути на кнопку “Відкрити посилання” відповідного елементу списку результату. Після цього посилання буде відкрито у окремій вкладинці поточного браузеру.

Слід відзначити, що систему працює і з англомовними текстами, як показано на рисунку 5.7:

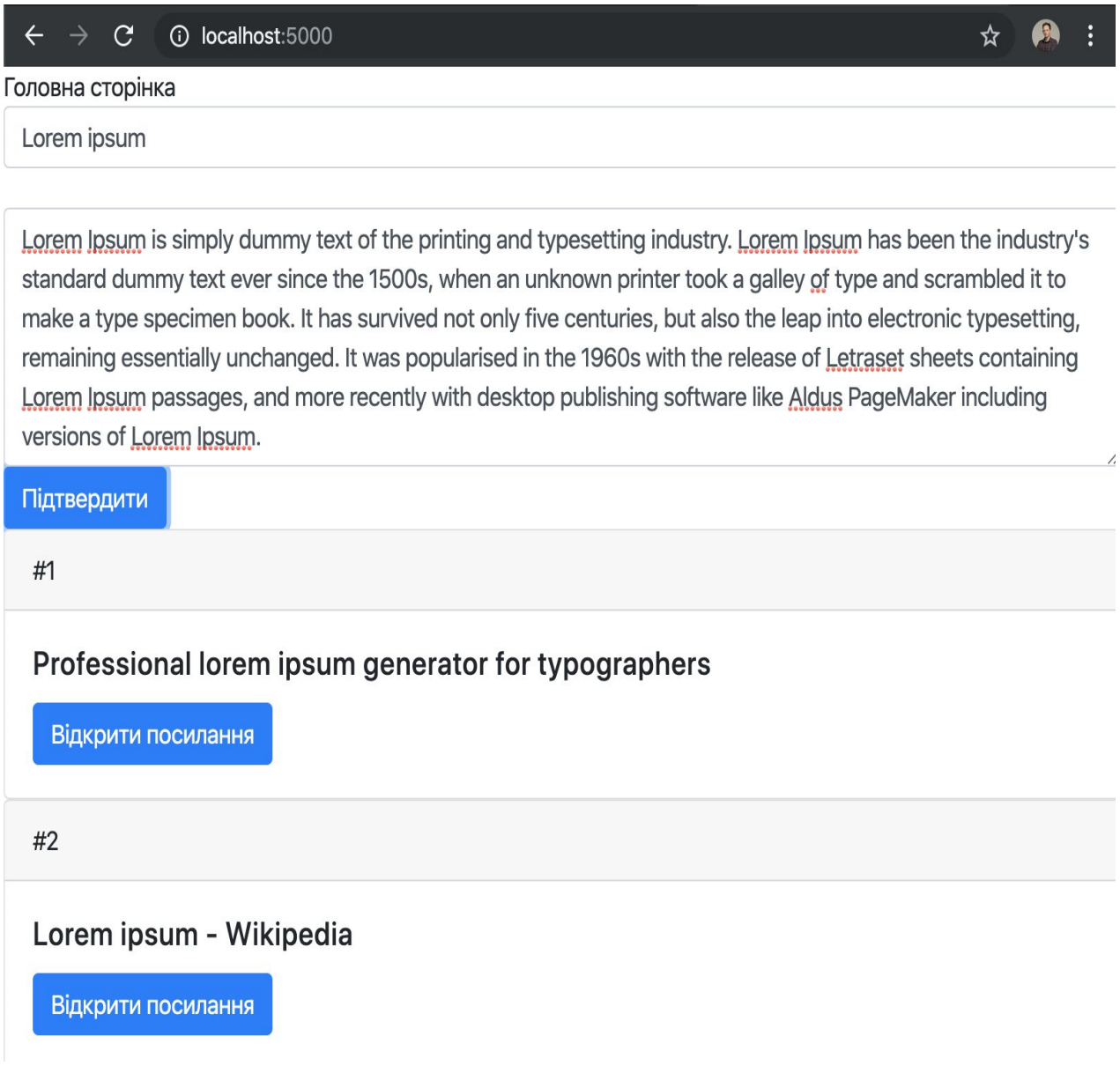

Рисунок 5.7 — Результат перевірки англомовного тексту

Висновок до розділу 5

У даному розділі було описано можливі сценарії взаємодії користувача з додатком. Були описані та наглядно продемонстровані основні можливості

програмного продукту, такі як запуск системи, роботу з формою аналізу текстової інформації. Були описані основні вимоги до інсталяції та рекомендації щодо використання програмного продукту.

6. РОЗРОБЛЕННЯ СТАРТАП ПРОЕКТУ

Стартап як форма малого ризикового підприємництва впродовж останніх десяти років набула широкого розповсюдження у світі через зниження бар'єрів входу в ринок (з появою Інтернету як інструменту комунікацій та збуту стало значно простіше знаходити споживачів та інвесторів, займатись пошуком ресурсів, перетинати кордони між ринками різних країн), і вважається однією із складових інноваційної економіки, оскільки за рахунок мобільності, гнучкості та великої кількості стартап-проектів загальна маса інноваційних ідей зростає.

6.1. Опис ідеї проекту

Основною метою даної дисертації є розробка системи аналізу збіжності текстової інформації для оцінки плагіату з використанням модифікованого векторного алгоритму порівняння текстової інформації. Опис представлено на таблиці 6.1:

Таблиця 6.1 — Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Розробка системи аналізу збіжності текстової інформації для оцінки плагіату з використанням модифікації	1. Пошук джерел інформації, з які мають деяку подібність із фрагментом, що перевіряється	1. Перевірка фрагменту текстової інформації на предмет запозичення з інших джерел.
		2. Перевірка частоти використання фрагменту в інших джерелах

Проведено аналіз потенційних техніко-економічних властивостей та характеристик ідеї та визначено орієнтовне коло конкурентів, як показано на таблиці 6.2, W - слабка сторона, N - нейтральна сторона, а S - сильна сторона ідеї:

Таблиця 6.2 — Аналіз техніко-економічних властивостей ідеї

№ п/п	Техніко-економічні характеристики ідеї	товари/концепції конкурентів				W	N	S
		Мій проект	Viper	Unicheck	Advego plagiatus			
1	Відкритий вихідний код	Є доступним на Github	Ні	Ні	Ні	Конкуренти можуть дослідити особливості реалізації	Сприяння розвитку програмування шляхом розповсюдження великих проектів із відкритим вихідним кодом	Клієнти завжди зможуть перевірити продукт так як код є відкритим
2	Точність роботи алгоритму	Використовується векторний метод із токенами триграм	Не відомо	Не відомо	Не відомо	Добре перевіряти лише фрагменти	Вектори після створення можна перевіряти різними алгоритмами	Алгоритм є дуже чутливим до сполучень слів, тому однозначно визначає схожі тексти

Було визначено технологічну здійсненність ідеї проекту у таблиці 6.3:

Таблиця 6.3 — Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технологія її реалізації	Наявність технологій	Доступність технологій
1	Обрати платформу для вирішення задачі	Клієнт-серверний додаток	Веб-сервери на Python, Node.js, Java, C# і ще багато іншого.	Доступні
		Мобільний додаток	Додатки на Java, Kotlin, Swift, React native.	Доступні
		Додаток для ПК	Додатки на Python, Node.js, Java, C# і ще багато іншого.	Доступні
2	Обрати бібліотеку для пошуку інформації у відкритих джерелах	Python модуль що робить віддалені запити	Python бібліотека googlesearch	Доступна безкоштовно
3	Обрати бібліотеку для обробки текстової інформації і пошуку подібності	Python модуль, що надає функцію для порівняння текстів	Python бібліотека sklearn	Доступна безкоштовно
4	Розробити систему для перевірки алгоритму	веб-сервер на Python	веб-фреймворк Flask	Доступна безкоштовно
		веб-додаток на Javascript	бібліотека React	Доступна безкоштовно під ліцензією BSD
		база даних для збереження стану системи	MongoDB	Безкоштовна Community версія
5	Розробити модифікацію для векторного алгоритму	Python модуль, що надає функцію для порівняння текстів з використання триграм	Python бібліотека sklearn	Доступна безкоштовно

Проведена робота показує, що для ідея проекту є можливою. Модуль пошуку інформації у відкритих джерелах розміщується окремо від основних модулів системи для того, щоб у подальшому можна було використати будь-яке джерело пошуку відкритих джерел. Все буде залежати від програмної реалізації, але до системи можна буде підключити будь-яку електронну бібліотеку, або ж пошукову систему. Було обрано реалізацію ідеї у вигляді клієнт-серверного додатку з використанням Python модулів. Такий вибір надає можливість використовувати велику кількість аналітичних, потужних модулів для обробки великої кількості інформації. Це дозволило у досить короткий термін розробити першу доступну програмну реалізацію. Саме перша проста програмна реалізація дозволила оцінити подальші перспективи розвитку проекту. Тобто, спочатку було розроблено модуль який приймав як аргументи тексти для перевірки і повертав коефіцієнти подібності. Робив він це використовуючи вже написані функції, які надає бібліотека sklearn.

Для пошуку інформації для перевірки фрагменту було розроблено модуль пошуку інформації у відкритих джерелах. Модуль googlesearch цей етап значно полегшив і дозволив робити запити у Google.

Для наглядності роботи алгоритму було розроблено систему, яка дозволяла ввести фрагмент тексту для перевірки через браузер користувача. Запит від клієнта йшов до серверу який викликав модуль пошуку інформації у відкритих джерелах. Потім знайдені тексти передавалися модулю перевірки текстів і обраховані коефіцієнти поверталися клієнту. Ця система, власне, і дозволила провести аналіз точності використовуваного алгоритму. Після цього і було прийнято рішення про модифікацію використовуваного алгоритму.

Для модифікації алгоритму було знову ж таки використано бібліотеку sklearn, так як вона надавала такі можливості. Так як використовувався векторний метод порівняння, було вирішено модифікувати його додаванням перевірки токенів триграм.

Також було у таблиці 6.4 визначено ринкові можливості, які можна використати під час ринкового впровадження проекту, та ринкові загрози, які можуть перешкодити реалізації проекту. Все це дозволить спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Таблиця 6.4 — Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	10
2	Загальний обсяг продаж, грн/ум.од	20
3	Динаміка ринку (якісна оцінка)	зростає
4	Наявність обмежень для входу	немає
5	Специфічні вимоги до стандартизації та сертифікації	Купити домен та згенерувати HTTPS сертифікати
6	Середня норма рентабельності в галузі (або по ринку), %	0.5

Проведений аналіз показав, що проект не є дуже прибутковим, тому не є дуже привабливим для вкладень. Проте якщо розглядати систему із боку розробленої модифікації алгоритму, можливо є сенс вкладати гроші у розробку таких швидких і точних алгоритмів перевірки подібності текстів. Адже основною ідеєю даного проекту є перевірка життєздатності розробленого алгоритму а не самої системи. Проведена робота дозволяє зробити висновок, що алгоритм є життєздатним і може бути інтегрованим у більш складні системи перевірки інформації, які будуть ділити тексти на фрагменти і використовувати цей метод для перевірки самих фрагментів.

Слід відзначити легкість впровадження розробленої системи для роботи у навантаженому режимі. Потрібно розмістити додаток на сервері і купити домен із HTTPS сертифікатом.

Далі було визначено основні групи клієнтів, як показано на таблиці 6.5:

Таблиця 6.5 — Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
	Необхідно мати можливість перевірити фрагмент тексту на подібність із текстами, доступними у відкритих джерелах	Цільовою аудиторією проекту є наукові працівники, наукові структури, люди які публікують статті, виступають на конференціях, студенти університетів, структури які приймають і публікують статті, викладачі	Відсутня	Єдиною вимогою до користувачів до товару є або реєстрування у доступному додатку, або ж встановлення системи на власних серверах

Клієнтами даної системи можуть бути різні люди, які приймають участь у публікації статей, організації конференцій, роботи з науковим і не тільки матеріалом.

Для того щоб користуватися даною систему, користувачу необхідно буде зареєструватися у системі. Після цього він зможе заходити до свого електронного кабінету і перевіряти фрагменти тексту на подібність із відкритими джерелами. Якщо дуже необхідно, користувач за окрему плату може запросити встановлення системи на його серверах.

Після визначення потенційних груп клієнтів було проведено аналіз ринкового середовища. Фактори загроз представлено у таблиці 6.6:

Таблиця 6.6 — Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Устарювання алгоритму	З часом алгоритм може втратити свою ефективність у порівнянні з іншими присутніми на ринку	Вкласти часові ресурси у розроблення більш ефективного алгоритму
2	Недоступність серверів	Може статися що додаток може бути недоступним внаслідок проблем провайдера чи перевантаження	Збільшити ресурси для серверів, додати захист від DDOS атак.

Також були проаналізовані основні фактори можливостей у таблиці 6.7:

Таблиця 6.7 — Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Додання можливості перевірки великих текстів	Якщо система стане успішно аналізувати великі тексти, а не лише фрагменти, то буде значно перевіряти великі статті	Розробити алгоритм розбиття великого тексту на фрагменти, які потім можна успішно проаналізувати.
2	Додання можливості періодичної перевірки заданих відкритих джерел	Було б дуже зручно налаштувати систему на періодичну обробку відкритих джерел для постійної підтримки метаінформації про статтю, або ж відстеження у яких джерелах використовуються ті чи інші джерела	Розробити модуль для виконання періодичної перевірки відкритого джерела за посиланням і збереження цієї інформації у БД.

Проведений аналіз дозволяє зробити висновок, що існує немало шляхів для вдосконалення поточної системи. Існують фактори загроз до яких необхідно бути готовим при виході на стартап ринок. Також існують можливості, які дозволять зробити декілька впевнених кроків у засвоєнні ринку.

Далі було проведено аналіз пропозиції: визначено загальні риси конкуренції, як показано у таблиці 6.8:

Таблиця 6.8 — Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства
1. Конкуренція чиста	Усі конкуренти займають мінімально різні ніші на ринку. Хтось працює із науковими статтями, хтось науково-популярними чи технічними.	Система може спробувати зайняти одразу багато ніш, так як не прив’язана до конкретної галузі
2. Рівень конкурентної боротьби - локальний	Конкуренти в основному працюють на багатьох ринках і тому не багато з них працює лише на ринку своєї країни	Знову ж таки для системи не є таким важливим фактором мова. Головне розробити інтерфейс для роботи з даною мовою, а сам алгоритм працює уже просто з символами.
3. За галузевою ознакою - внутрішньогалузева	Кожна система добре налаштована для роботи із текстами одного типу (наукові, технічні, публіцистика)	Система може бути налаштована на різні системи пошуку інформації у відкритих джерелах. Тобто, якщо додати модуль, який буде працювати із науковими статтями, то система буде працювати із науковими статтями
4. Конкуренція за видами товару - товарно-видова	Конкуренція утворюється між системами гарно працюючими в одній галузі	Система із відповідним модулем пошуку інформації повинна бути гарно відтестована і перевірена
5. За характером конкурентних переваг - цінова	Конкуренція утворюється і на основі ціни на послуги	На менш детальну перевірку тексту можна робити меншу ціну
6. За інтенсивністю - не немарочна	Не так важлива яка компанія випустила той чи інший продукт - головне наскільки гарно він виконує свої функції	Система повинна бути гарно відладженою і з приємним інтерфейсом

Після аналізу конкуренції було проведено більш детальний аналіз умов конкуренції галузі

Таблиця 6.9 — Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Viper, Unicheck, Advego Plagiatus	Входження у ринок обмежене лише часом налаштування робочого середовища системи	Провайдером робочого середовища може бути AWS або ж GCP	Споживачі будуть раді виходу на ринок дешевої системи перевірки на плагіат	Замінниками можут бути ти ж самі конкуренти
Висновки:	Конкуренція є дуже високою, але у деяких випадках переваги полягають у непретендування на права статей користувачі або ж низька ціна входу, або ж більша доступність	Вихід на ринок не є дуже складною задачею, необхідно налаштувати робоче середовище, а це може бути виконано за тиждень	Так, провайдери серверів звичайно диктують свої умови, наприклад ціна забезпечення серверів. Проте ці провайдери відомі як дуже надійні і вони забезпечують інструменти для забезпечення високої доступності системи і її потужності	Так, звичайно і клієнти диктують умови. Наприклад, якщо випустити систему, яка робить те, що нікому не потрібно, то на цій системі і не вийде нічого заробити. Тому система повинна реалізувати функціонал, який є необхідним для користувачів і за який вони готові платити	Товарі замінників не передбачено

Проведений аналіз дозволяє зробити висновок, що на ринку представлена не слабка конкуренція. Проте якщо всі аспекти системи будуть виконані так як заплановано, то це дасть системі значну перевагу перед конкурентами. Так наприклад, якщо буде гарно реалізовано модуль пошуку інформації у відкритих джерелах і він буде легко замінятися іншими пошуковими системами чи навіть

електронними бібліотеками. Така модифікація дозволить розробити такі собі продукти, які будуть працювати на різних нішах ринку і розповсюджувати вплив самої системи на всьому ринку.

На основі аналізу конкуренції, а також із урахуванням характеристик ідеї проекту, вимог споживачів до товару та факторів маркетингового середовища у таблиці 6.10 було визначено перелік факторів конкурентноспроможності.

Таблиця 6.10 — Обгрунтування факторів конкурентноспроможності

№ п/п	Фактор конкурентноспроможності	Обгрунтування
1	Векторний алгоритм з використанням триграм	Використовується контекстно-незалежний метод для оцінки текстової інформації. Метод не залежить від мови, є удосконаленим за допомогою триграм, що робить його точнішим у визначенні запозичених фрагментів тексту, і менш чутливим до омонімів чи співпадіння деякої кількості слів.
2	Окремий модуль пошуку інформації у відкритих джерелах	Використовується окремий модуль пошуку інформації у відкритих джерелах, що робить можливим у майбутньому використання будь якої системи пошуку інформації і налаштування системи під різні середовища. Наприклад, якщо система пошуку інформації буде налаштована на наукові статті, то система буде гарно відстежувати подібні тексти у цій галузі
3	Клієнт-серверна архітектура	Використовується підхід клієнт-серверної архітектури що дозволяє окремо підвищувати потужність серверу, незалежно від клієнтської частини і економити ресурси

У таблиці 6.11 було проведено аналіз сильних і слабких сторін стартап-проекту:

Таблиця 6.11 — Порівняльний аналіз сильних та слабких сторін “Система аналізу збіжності текстової інформації для оцінки плагіату”

№ п/п	Фактор конкурентноспроможності	Бали	Рейтинг товарів конкурентів у порівнянні з Viper						
			-3	-2	-1	0	1	2	3
1	Векторний алгоритм з використанням триграм	13				+			
2	Окремий модуль пошуку інформації у відкритих джерелах	19							+
3	Клієнт-серверна архітектура	15						+	

На основі проведеного аналізу було складено SWOT-аналіз проекту у таблиці 6.12:

Таблиця 6.12 — SWOT-аналіз стартап-проекту

Сильні сторони: векторний метод аналізу текстової інформації з використанням триграм, клієнт-серверна архітектура, окремий модуль пошуку інформації у відкритих джерелах	Слабкі сторони: висока точність аналізу гарантована лише на фрагментах тексту, малий функціонал, застарілий інтерфейс користувача
Можливості: використання різних модулів пошуку інформації у відкритих джерелах, оновлення інтерфейсу користувача, обробку великих текстів із розбиттям на фрагменти	Загрози: DDOS атаки, висока вартість забезпечення серверів

На основі SWOT-аналізу було розроблено альтернативи ринкової поведінки для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації. Результат наведено у таблиці 6.13:

Таблиця 6.13 — Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива	Ймовірність отримання ресурсів	Строки реалізації
1	Безкоштовне бета-тестування для наукових працівників	Велика	Одразу

2	Надання можливості обробки Microsoft Word документів	Велика	Необхідно витратити час на розробку від тижня до місяця

Для розроблення ринкової стратегії першим кроком є визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів показано у таблиці 6.14.

Таблиця 6.14 — Вибір цільових груп потенційних користувачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Науковці	Дуже необхідно мати інструмент для оцінки оригінальності ідеї у тому чи іншому джерелі	Більшість тих хто ще не є клієнтом конкуренту	Конкуренція не є дуже інтенсивною	Необхідно замінити модуль пошуку інформації на такий, який буде працювати із науковими джерелами
2	Викладачі	Можна перевіряти роботи студентів на оригінальність	Не дуже високий, адже зазвичай спеціаліст може легко відрізнити оригінальну думку від плагіату	Конкуренція не є дуже інтенсивною	Немає перешкод
Які цільові групи обрано: науковці і викладачі					

Після проведеного аналізу можна зробити висновок про основні цільові групи, але система добре підійде і для людей, які займаються випуском та перевіркою статей. Для того ж щоб працювати на різні цільові групи, необхідно буде реалізувати різні модулі пошуку інформації у відкритих джерелах.

Далі було сформовано базову стратегію розвитку, як показано у таблиці 6.15:

Таблиця 6.15 — Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект “першопрохідцем” на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів,	Чи буде компанія копіювати основні характеристики товару конкуренту, і які?	Стратегія конкурентної поведінки
	Ні	Метою є популяризація систем аналізу текстової інформації. Тобто планується пошук нових користувачів із тих хто раніше ніколи не використовував таке ПЗ. Проте якістю і меншою ціною можна буде зазіхнути на користувачів конкурентів	Можливо копіювати якісь елементи інтерфейсу користувача, що покращує задоволення від користування системою	Стратегія виклику лідера

На основі вимог споживачів з обраних сегментів до постачальника та до продукту, а також в залежності від обраної стратегії розвитку та стратегії конкурентної поведінки була розроблена стратегія позиціонування, що полягає у формуванні ринкової позиції, за яким споживачі мають ідентифікувати торгівельний проект.

Результатом проведення даного аналізу стала узгоджена система рішень щодо ринкової поведінки стартап-компанії, яка визначатиме напрями роботи стартап-компанії на ринку.

Далі необхідно було сформувати маркетингову концепцію потенційного товару. Для цього було створено табличку 6.16:

Таблиця 6.16 — Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує	Ключові переваги перед конкурентами

п/п		товар	
1	Перевірка фрагменту текстової інформації для оцінки плагіату	Надає можливість перевірку фрагменту текстової інформації для оцінки на плагіат	Перевагою є використання векторного алгоритму модифікованого за допомогою токенів триграм, що робить його контекстно-незалежним і більш точним у пошуку подібності текстової інформації
2	Налаштування на періодичну перевірку джерела інформації на використання у інших джерелах	Може надавати таку можливість	Перевагою є відсутність таких характеристик у конкурентів

Висновок до розділу 6

Проведений аналіз дозволяє зробити висновок, що є можливість ринкової комерціалізації проекту. На ринку наявний попит на дані послуги, конкуренція не є надто щільною та існує функціонал, якого потребує користувач, але поки ніхто не може його надати. Подальша імплементація проекту є доцільною.

ВИСНОВКИ

Тема розробки системи аналізу збіжності текстової інформації для аналізу плагіату є досить цікавою та представляє широке поле для подальших досліджень.

Метою даної роботи було використання архітектури клієнт-сервер для створення програмного продукту, який представляє собою систему аналізу збіжності текстової інформації для оцінки плагіату, використовуючи модифікований алгоритм векторного аналізу.

Інструментарій для виконання роботи було обрано з огляду на його функціональність та відповідність нашим вимогам для даної роботи. Фреймворк Flask дозволив створити серверну частину, що оперує з БД MongoDB та модулем sklearn, а також за допомогою зручної мови програмування Python реалізувати логіку серверу. Редактор Visual Studio Code дозволив писати код програми, і зручно проводити процес відлагодження та пошуку помилок у програмі. Бібліотека React дозволила реалізувати інтерактивний користувацький інтерфейс, а jQuery полегшила роботу з окремими елементами дерева DOM. Інструмент Webpack дозволив перетворити високорівневий код, написаний за допомогою React, і перетворив його у зменшений крос-браузерний JavaScript код.

Проведений аналіз показав, що за допомогою програмного середовища Python [1] (а саме Flask, sklearn, numpy) можна розробити програмний продукт для пошуку і порівняння інформації з відкритих джерел. У якості алгоритму для оцінки збіжності було вирішено використати векторний метод із групуванням слів у тріади (3-грами) [2]. Це дозволило відстежувати не тільки ідентичні фрагменти тексту з інших джерел, а і мінімально трансформовані речення, у яких, наприклад, слова переставлені місцями. В той же час, такий алгоритм менш чутливий до омонімів і текстів зі схожим набором слів, але зовсім іншим сенсом. Варте уваги і те, що така модифікація алгоритму є більш складною задачею з точки зору обчислень, тому що

тепер потрібно комбінувати слова у групи з по 3 елементи і зі збільшенням кількості слів збільшується і розмір вектору, а звідси збільшується час виконання алгоритму.

Розроблена система, отримавши запит на перевірку текста, починає пошук заданої теми у відкритих джерелах і завантажує тексту сторінок. Після цього відбувається виокремлення текстової інформації з HTML-коду і створення вектору. Вектори всіх знайдених джерел перевіряються на подібність і відбираються найбільш схожі методом косинусу подібності двох векторів [2]. Після цього система відображає користувачеві список посилань на джерела з яких ймовірно взята інформація у тексті, що перевіряється. Розроблений підхід досить ефективно проводить перевірку тексту на плагіат або навіть пошук робіт, які були використані у тексті, що перевіряється.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Амосов А. А. Вычислительные методы для инженеров / А. А. Амосов, Ю. А. Дубинский, Н. П. Копченова. — М: Мир, 1998. – 644 с.

2. Бахвалов Н. С. Численные методы / Н. С. Бахвалов, Н. П. Жидков, Г. М. Кобельков. — Москва: Бином, 2001. — 363 с.

3. Бен Сміт «Beginning JSON» / Бен Сміт – К. : «Apress», 2015. – 324 с.

4. Боровков А. И. Компьютерный инжиниринг / А. И. Боровков. — СПб: Изд-во Политехн. ун-та, 2012. — 93 с.

5. Гаврилова Т.А. Базы знаний интеллектуальных систем: підруч. [для студ. вищ. навч. закл] / Гаврилова Т.А., Хорошевський В.Ф. – Санкт-Петербург : Питер, 2000. – 384 с.

6. Гольцман В.І. «MySQL 5.0. Библиотека программиста» / Гольцман В.І. – К.: «Питер», 2010. – 253 с.

7. Дейт К. Дж. Введение в системы баз данных — 8-е изд. / Дейт К.Дж. : «Вильямс», 2006. — 1328 ст.

8. Лутц М. Программирование на Python / Марк Лутц. — СПб.: Символ-Плюс, 2011. — 992 с.

9. Портнякин И. Эффективные пользовательские интерфейсы / Иван Портнякин. — Санкт-Петербург: Лори, 2011. — 600 с.

10. Робін Ніксон «Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 3-е издание» / Робін Ніксон – К. : «Питер», 2015.– 688 с.

11. M. Fowler. Patterns of Enterprise Application Architecture / M.Fowler, D.Rice, M.Foemmel – Addison Wesley, 2002 – 560 с.

12. Современный учебник Javascript [Електронний ресурс] — Режим доступу: https://learn.javascript.ru/

13. Chrome DevTools [Електронний ресурс] — Режим доступу: https://developers.google.com/web/tools/chrome-devtools/

14. Client-Server Interaction over the Internet – Режим доступу: http://www.icodeguru.com/dotnet/core.c.sharp.and.dot.net/0131472275/ch16le1sec1.html. – Дата доступу: 12.05.2015.

15. Flask [Електронний ресурс] — Режим доступу: https://palletsprojects.com/p/flask/

16. jQuery [Електронний ресурс] — Режим доступу: https://jquery.com/

17. LINQ to SQL Execution Architecture – Режим доступу: http://blogs.msdn.com/b/wriju/archive/2007/12/18/linq-to-sql-executionarchitecture.aspx. - Дата доступу: 18.05.2015.

18. MongoDB Documentation [Електронний ресурс] — Режим доступу: https://docs.mongodb.com/

19. REST Principles and Architectural Constraints – REST API Tutorial [Електронний ресурс] — Режим доступу: https://restfulapi.net/rest-architectural-constraints/

20. Node.js [Електронний ресурс] — Режим доступу: https://nodejs.org/

21. React.js [Електронний ресурс] — Режим доступу: https://reactjs.org/

22. Visual Studio Code [Електронний ресурс] — Режим доступу: https://www.code.visualstudio.co/

23. Webpack [Електронний ресурс] — Режим доступу: https://webpack.js.org/

24. Плагіат [Електронний ресурс] — Режим доступу: https://uk.wikipedia.org/wiki/Плагіат

25. scikit-learn: machine learning in Python [Електронний ресурс] — Режим доступу: https://scikit-learn.org/stable/

26. Text Similarities: Estimate the degree of similarity between two texts [Електронний ресурс] — Режим доступу: https://medium.com/@adriensieg/text-similarities-da019229c894

27. Git [Електронний ресурс] — Режим доступу: https://git-scm.com

28. Beautiful Soup Documenation [Електронний ресурс] — Режим доступу: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Додаток А

Система аналізу збіжності текстової інформації для оцінки плагіату

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР4157_19М

Аркушів 4

Київ 2019


www.konferenciaonline.org.ua
Міжнародна наукова
інтернет-конференція
Інформаційне суспільство:
технологічні, економічні
та технічні аспекти становлення
(випуск 43)
Частина 1
ISSN 2522-932X
14 листопада 2019 р.
Тернопіль
2019

Кузьміних В.О., канд. тех. наук, доцент
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського", м. Київ
Кафедра автоматизації проектування енергетичних процесів і систем, доцент
Пивовар Н.О., студент ТВ-з81мп
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського", м. Київ
Кафедра автоматизації проектування енергетичних процесів і систем, студент

СИСТЕМА АНАЛІЗУ ЗБІЖНОСТІ ТЕКСТОВОЇ ІНФОРМАЦІЇ ДЛЯ ОЦІНКИ ПЛАГІАТУ

Розвиток та розповсюдження засобів комунікації і доступу до мережі Інтернет сприяє збільшенню спектру інформації до якої людина має доступ. За допомогою звичайного смартфону можна здійснити пошук будь-якої інформації. Тим не менш, звідси виникає нова проблема: так як кожен може отримати доступ до будь-якої інформації, значно складніше оцінити чи є вона оригіналом чи взята з інших відкритих джерел. Досить часто у мережі Інтернет можна знайти джерела інформації (будь то статті, сайти, онлайн журнали і т.п.), які майже повністю складаються с фрагментів тексту інших джерел. Сьогодні існують такі програми як Advego Plagiatius, Etxt Antiplagiat та онлайн сервіси як UNPLAG, Content-Watch та інші. Проте мало які системи дають точні результати і рекомендується перевіряти результати у декількох системах. Так як для людини це досить не проста задача було прийнято рішення розробки системи аналізу збіжності текстової інформації для оцінки плагіату.

Проведений аналіз показав, що за допомогою програмного середовища Python [1] (а саме Flask, sklearn, numpy) можна розробити програмний продукт для пошуку і порівняння інформації з відкритих джерел. У якості алгоритму для оцінки збіжності було вирішено використати векторний метод із групуванням слів у тріади (3-грами) [2]. Це дозволило відстежувати не тільки ідентичні фрагменти тексту з інших джерел, а і мінімально трансформовані речення, у яких, наприклад, слова переставлені місцями. В той же час, такий алгоритм менш чутливий до омонімів і текстів зі схожим набором слів, але зовсім іншим сенсом. Варте уваги і те, що така модифікація алгоритму є більш складною задачею з точки зору обчислень, тому що тепер потрібно комбінувати слова у групи з по 3 елементи і зі збільшенням кількості слів збільшується і розмір вектору, а звідси збільшується час виконання алгоритму.

Розроблена система, отримавши запит на перевірку текста, починає пошук заданої теми у відкритих джерелах і завантажує тексту сторінок. Після цього відбувається виокремлення текстової інформації з HTML-коду і створення вектору. Вектори всіх знайдених джерел перевіряються на подібність і відбираються найбільш схожі методом косинусу подібності двох векторів [2]. Після цього система відображає користувачеві список посилань на джерела з

яких ймовірно взята інформація у тексті, що перевіряється. Блок-схема алгоритму зображено на рисунку 1.1.

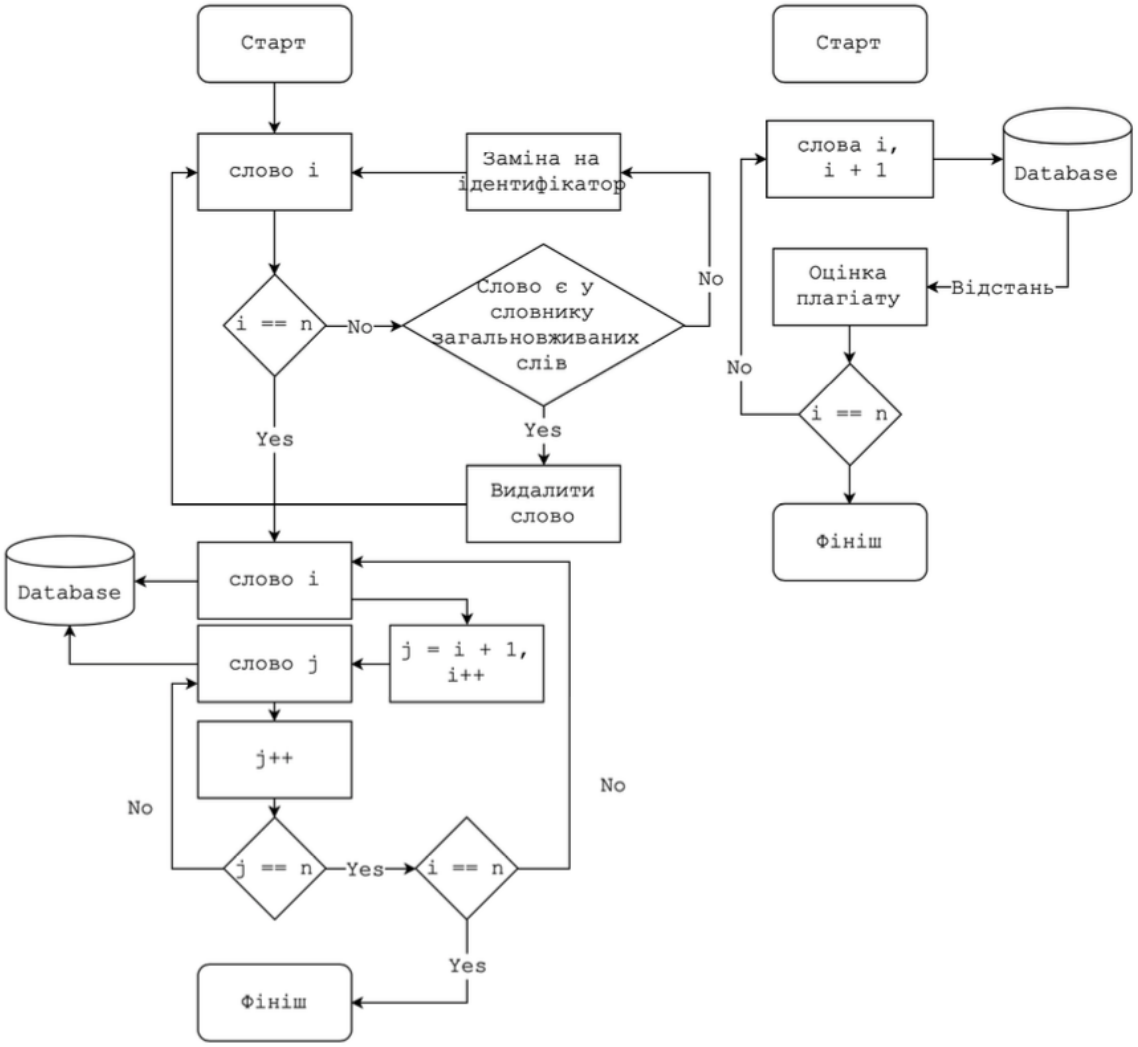

Рисунок 1.1

Розроблений підхід досить ефективно проводить перевірку тексту на плагіат або навіть пошук робіт, які були використані у тексті, що перевіряється.

Література:

1. Марк Лутц. Программирование на Python / Марк Лутц. — СПб.: Символ-Плюс, 2011. — 992 с.
2. sklearn.metrics.pairwise.cosine_similarity — scikit-learn 0.21.3 documentation [Електронний ресурс]: — Режим доступу: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html.